[MS-TDS]:
Tabular Data Stream Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Copyright © 2014 Microsoft Corporation.

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2008	0.1	Major	Initial Availability.
06/20/2008	0.1.1	Editorial	Revised and edited the technical content.
07/25/2008	0.1.2	Editorial	Revised and edited the technical content.
08/29/2008	0.1.3	Editorial	Revised and edited the technical content.
10/24/2008	0.1.4	Editorial	Revised and edited the technical content.
12/05/2008	0.2	Minor	Updated the technical content.
01/16/2009	0.3	Minor	Updated the technical content.
02/27/2009	0.4	Minor	Updated the technical content.
04/10/2009	0.5	Minor	Updated the technical content.
05/22/2009	0.5.1	Editorial	Revised and edited the technical content.
07/02/2009	1.0	Major	Updated and revised the technical content.
08/14/2009	1.1	Minor	Updated the technical content.
09/25/2009	2.0	Major	Updated and revised the technical content.
11/06/2009	3.0	Major	Updated and revised the technical content.
12/18/2009	4.0	Major	Updated and revised the technical content.
01/29/2010	4.1	Minor	Updated the technical content.
03/12/2010	5.0	Major	Updated and revised the technical content.
04/23/2010	6.0	Major	Updated and revised the technical content.
06/04/2010	7.0	Major	Updated and revised the technical content.
07/16/2010	8.0	Major	Significantly changed the technical content.
08/27/2010	8.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	9.0	Major	Significantly changed the technical content.
11/19/2010	9.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	9.1	Minor	Clarified the meaning of the technical content.
02/11/2011	9.2	Minor	Clarified the meaning of the technical content.

Copyright © 2014 Microsoft Corporation.

Date	Revision History	Revision Class	Comments
03/25/2011	9.3	Minor	Clarified the meaning of the technical content.
05/06/2011	9.4	Minor	Clarified the meaning of the technical content.
06/17/2011	10.0	Major	Significantly changed the technical content.
09/23/2011	11.0	Major	Significantly changed the technical content.
12/16/2011	12.0	Major	Significantly changed the technical content.
03/30/2012	12.1	Minor	Clarified the meaning of the technical content.
07/12/2012	12.2	Minor	Clarified the meaning of the technical content.
10/25/2012	12.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	13.0	Major	Significantly changed the technical content.
08/08/2013	14.0	Major	Significantly changed the technical content.
11/14/2013	15.0	Major	Significantly changed the technical content.
02/13/2014	16.0	Major	Significantly changed the technical content.
05/15/2014	17.0	Major	Significantly changed the technical content.

Copyright © 2014 Microsoft Corporation.

Contents

1 Introduction 8
1.1 Glossary 8
1.2 References 10
1.2.1 Normative References 10
1.2.2 Informative References 11
1.3 Overview 12
1.4 Relationship to Other Protocols 14
1.5 Prerequisites/Preconditions 14
1.6 Applicability Statement 15
1.7 Versioning and Capability Negotiation 15
1.8 Vendor-Extensible Fields 15
1.9 Standards Assignments 15

2 Messages 16
2.1 Transport 16
2.2 Message Syntax 16
2.2.1 Client Messages 16
2.2.1.1 Pre-Login 17
2.2.1.2 Login 17
2.2.1.3 SQL Batch 17
2.2.1.4 Bulk Load 17
2.2.1.5 Remote Procedure Call 17
2.2.1.6 Attention 17
2.2.1.7 Transaction Manager Request 18
2.2.2 Server Messages 18
2.2.2.1 Pre-Login Response 18
2.2.2.2 Login Response 18
2.2.2.3 Row Data 19
2.2.2.4 Return Status 19
2.2.2.5 Return Parameters 19
2.2.2.6 Response Completion ("DONE") 19
2.2.2.7 ERROR and INFO Messages 19
2.2.2.8 Attention Acknowledgment 20
2.2.3 Packets 20
2.2.3.1 Packet Header 20
2.2.3.1.1 Type 20
2.2.3.1.2 Status 22
2.2.3.1.3 Length 22
2.2.3.1.4 SPID 22
2.2.3.1.5 PacketID 23
2.2.3.1.6 Window 23
2.2.3.2 Packet Data 23
2.2.4 Packet Data Token and Tokenless Data Streams 23
2.2.4.1 Tokenless Stream 24
2.2.4.2 Token Stream 24
2.2.4.2.1 Token Definition 24
2.2.4.2.1.1 Zero Length Token(xx01xxxx) 24
2.2.4.2.1.2 Fixed Length Token(xx11xxxx) 25
2.2.4.2.1.3 Variable Length Tokens(xx10xxxx) 25
2.2.4.2.1.4 Variable Count Tokens(xx00xxxx) 26

Copyright © 2014 Microsoft Corporation.

2.2.4.3 Done and Attention Tokens 26
2.2.5 Grammar Definition for Token Description 26
2.2.5.1 General Rules 26
2.2.5.1.1 Least Significant Bit Order 29
2.2.5.1.2 Collation Rule Definition 29
2.2.5.2 Data Stream Types 30
2.2.5.2.1 Unknown Length Data Streams 30
2.2.5.2.2 Variable-Length Data Streams 30
2.2.5.2.3 Data Type Dependent Data Streams 31
2.2.5.3 Packet Data Stream Headers - ALL_HEADERS Rule Definition 32
2.2.5.3.1 Query Notifications Header 33
2.2.5.3.2 Transaction Descriptor Header 34
2.2.5.3.3 Trace Activity Header 34
2.2.5.4 Data Type Definitions 35
2.2.5.4.1 Fixed-Length Data Types 35
2.2.5.4.2 Variable-Length Data Types 36
2.2.5.4.3 Partially Length-Prefixed Data Types 39
2.2.5.5 Data Type Details 39
2.2.5.5.1 System Data Type Values 39
2.2.5.5.1.1 Integers 39
2.2.5.5.1.2 Timestamp 40
2.2.5.5.1.3 Character and Binary Strings 40
2.2.5.5.1.4 Fixed-Point Numbers 40
2.2.5.5.1.5 Floating-Point Numbers 40
2.2.5.5.1.6 Decimal/Numeric 40
2.2.5.5.1.7 GUID 41
2.2.5.5.1.8 Date/Times 41
2.2.5.5.2 Common Language Runtime (CLR) Instances 41
2.2.5.5.3 XML Values 42
2.2.5.5.4 SQL_VARIANT Values 42
2.2.5.5.5 Table Valued Parameter (TVP) Values 43
2.2.5.5.5.1 Metadata 43
2.2.5.5.5.2 Optional Metadata Tokens 46
2.2.5.5.5.3 TDS Type Restrictions 48
2.2.5.6 Type Info Rule Definition 49
2.2.5.7 Data Buffer Stream Tokens 50
2.2.6 Packet Header Message Type Stream Definition 51
2.2.6.1 Bulk Load BCP 51
2.2.6.2 Bulk Load Update Text/Write Text 51
2.2.6.3 LOGIN7 52
2.2.6.4 PRELOGIN 64
2.2.6.5 RPC Request 68
2.2.6.6 SQLBatch 70
2.2.6.7 SSPI Message 71
2.2.6.8 Transaction Manager Request 72
2.2.7 Packet Data Token Stream Definition 76
2.2.7.1 ALTMETADATA 76
2.2.7.2 ALTROW 79
2.2.7.3 COLINFO 80
2.2.7.4 COLMETADATA 81
2.2.7.5 DONE 83
2.2.7.6 DONEINPROC 85
2.2.7.7 DONEPROC 86

Copyright © 2014 Microsoft Corporation.

2.2.7.8 ENVCHANGE 87
2.2.7.9 ERROR 91
2.2.7.10 FEATUREEXTACK 94
2.2.7.11 INFO 96
2.2.7.12 LOGINACK 97
2.2.7.13 NBCROW 98
2.2.7.14 OFFSET 100
2.2.7.15 ORDER 101
2.2.7.16 RETURNSTATUS 102
2.2.7.17 RETURNVALUE 102
2.2.7.18 ROW 105
2.2.7.19 SESSIONSTATE 106
2.2.7.20 SSPI 107
2.2.7.21 TABNAME 108
2.2.7.22 TVP ROW 109

3 Protocol Details 111
3.1 Common Details 111
3.1.1 Abstract Data Model 111
3.1.2 Timers 111
3.1.3 Initialization 111
3.1.4 Higher-Layer Triggered Events 111
3.1.5 Message Processing Events and Sequencing Rules 111
3.1.6 Timer Events 115
3.1.7 Other Local Events 115
3.2 Client Details 115
3.2.1 Abstract Data Model 116
3.2.2 Timers 117
3.2.3 Initialization 117
3.2.4 Higher-Layer Triggered Events 117
3.2.5 Message Processing Events and Sequencing Rules 119
3.2.5.1 Sent Initial PRELOGIN Packet State 119
3.2.5.2 Sent TLS/SSL Negotiation Packet State 120
3.2.5.3 Sent LOGIN7 Record with Complete Authentication Token State 120
3.2.5.4 Sent LOGIN7 Record with SPNEGO Packet State 121
3.2.5.5 Logged In State 121
3.2.5.6 Sent Client Request State 122
3.2.5.7 Sent Attention State 122
3.2.5.8 Routing Completed State 122
3.2.5.9 Final State 122
3.2.6 Timer Events 123
3.2.7 Other Local Events 123
3.3 Server Details 123
3.3.1 Abstract Data Model 124
3.3.2 Timers 125
3.3.3 Initialization 125
3.3.4 Higher-Layer Triggered Events 125
3.3.5 Message Processing Events and Sequencing Rules 125
3.3.5.1 Initial State 125
3.3.5.2 TLS/SSL Negotiation State 126
3.3.5.3 Login Ready State 126
3.3.5.4 SPNEGO Negotiation State 128
3.3.5.5 Logged In State 128

Copyright © 2014 Microsoft Corporation.

3.3.5.6 Client Request Execution State 129
3.3.5.7 Routing Completed State 129
3.3.5.8 Final State 129
3.3.6 Timer Events 129
3.3.7 Other Local Events 129

4 Protocol Examples 130
4.1 Pre-Login Request 130
4.2 Login Request 131
4.3 Login Request with Federated Authentication 134
4.4 Login Response 141
4.5 Login Response with Federated Authentication Feature Extension Acknowledgement 146
4.6 SQL Batch Client Request 150
4.7 SQL Batch Server Response 152
4.8 RPC Client Request 153
4.9 RPC Server Response 156
4.10 Attention Request 157
4.11 SSPI Message 157
4.12 SQL Command with Binary Data 158
4.13 Transaction Manager Request 160
4.14 TVP Insert Statement 161
4.15 SparseColumn Select Statement 163
4.16 FeatureExt with SessionRecovery Feature Data 169
4.17 FeatureExtAck with SessionRecovery Feature Data 174
4.18 Table Response with SessionState Token Data 181
4.19 Token Stream Communication 182
4.19.1 Sending a SQL Batch 182
4.19.2 Out-of-Band Attention Signal 183

5 Security 184
5.1 Security Considerations for Implementers 184

6 Appendix A: Product Behavior 185

7 Change Tracking 189

8 Index 191

Copyright © 2014 Microsoft Corporation.

1 Introduction

The Tabular Data Stream (TDS) protocol is an application layer request/response protocol that facilitates interaction with a database server and provides for:

- Authentication and channel encryption negotiation.
- Specification of requests in SQL (including Bulk Insert).
- Invocation of a **stored procedure** or user-defined function, also known as a **remote procedure call (RPC)**.
- The return of data.
- Transaction manager requests.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

big-endian
interface (2)
little-endian
nullable column
Security Support Provider Interface (SSPI)
Session Multiplex Protocol (SMUX)
stored procedure
table response
transaction manager

The following terms are specific to this document:

bulk insert: A method for efficiently populating the rows of a table from the **client** to the **server**.

client: A program that establishes connections for the purpose of sending requests.

column: A set of data composed of the same field from each row in a table.

Common Language Runtime User-Defined Type (CLR UDT): A data type created and defined by the user on a database **server** that supports SQL by using a .NET Framework common language runtime assembly.

data store: A repository for data.

data stream: A stream of data that corresponds to specific TDS semantics. A single data stream can represent an entire TDS message or only a specific, well-defined portion of a TDS message. A TDS data stream can span multiple network data packets.

delete: To remove a row from a table.

Copyright © 2014 Microsoft Corporation.

Distributed Transaction Coordinator (DTC): A service that coordinates transactions across multiple databases. For more information, see [MSDN-DTC].

final state: The application layer has finished communication and the lower layer connection should be disconnected.

initial state: A prerequisite for application layer communication. A lower layer channel which can provide reliable communication must be established.

insert: To add a row to a table.

Microsoft/Windows Data Access Components (MDAC/WDAC): With Microsoft/Windows Data Access Components (MDAC/WDAC), developers can connect to and use data from a wide variety of relational and nonrelational data sources. You can connect to many different data sources using Open Database Connectivity (ODBC), ActiveX Data Objects (ADO), or OLE DB. You can do this through providers and drivers that are built and shipped by Microsoft, or that are developed by various third parties. For more information, see [MSDN-MDAC].

Multiple Active Result Sets (MARS): A feature introduced in SQL Server 2005 that allows applications to have more than one pending request per connection. For more information, see [MSDN-MARS].

out-of-band: A type of event that happens outside of the standard sequence of events. Specifically, the idea that a signal or message can be sent during an unexpected time and will not cause any protocol parsing issues.

query: A character string expression sent to a **data store** that contains a set of operations that request data from the **data store**.

query notification: A feature introduced in SQL Server 2005 that allows the **client** to register for notification on changes to a given **query** result. For more information, see [MSDN-QUERYNOTE].

Remote Procedure Call (RPC): The direct invocation of a **stored procedure** or user-defined function on the **server**.

request: A TDS message initiated by a **client** and sent to a **server**.

response: A TDS message sent by a **server** to a **client** related to a previously issued request.

result set: A set of **data streams** representing the result of a **query**. A result set starts with a COLMETADATA token and ends with a DONE, DONEPROC, or DONEINPROC token.

server: An application program that accepts connections to service requests by sending back responses. Any program might be capable of being both a **client** and a **server**. Use of these terms refers only to the role being performed by the program for a particular connection rather than to the program's capabilities in general.

SQL Server Native Client (SNAC): SNAC contains the SQL Server ODBC driver and the SQL Server OLE DB provider in one native dynamic link library (DLL) supporting applications using native-code APIs (ODBC, OLE DB, and ADO) to Microsoft SQL Server. For more information, see [MSDN-SNAC].

SPNEGO: Simple and Protected GSS-API Negotiation as defined by [RFC4178]. This mechanism is used by SSPI for negotiation.

SQL batch: A set of SQL statements.

Copyright © 2014 Microsoft Corporation.

SQL Server User Authentication (SQLAUTH): An authentication mechanism used to support user accounts on a database **server** that supports SQL. The username and password of the user account are transmitted as part of the login message that the **client** sends to the **server**.

SQL statement: A character string expression in a language the **server** understands.

structurally invalid: A data stream that does not follow the header defined, the rule for the specific message type defined in section 2, or both.

TDS session: A successfully established communication over a period of time between a **client** and a **server** on which the Tabular Data Stream (TDS) protocol is used for message exchange.

Unicode: The set of characters as defined by [UNICODE] that are encoded in UCS-2.

update: An add, modify, or **delete** operation of one or more objects or attribute values.

Virtual Interface Architecture (VIA): A high-speed interconnect requiring special hardware and drivers provided by third parties.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[IANAPORT] IANA, "Service Name and Transport Protocol Port Number Registry", http://www.iana.org/assignments/port-numbers

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://ieeexplore.ieee.org/servlet/opac?punumber=2355

[MS-BINXML] Microsoft Corporation, "SQL Server Binary XML Structure".

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989, http://www.ietf.org/rfc/rfc1122.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

Copyright © 2014 Microsoft Corporation.

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC4234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[RFC6234] Eastlake III, D., and Hansen, T., "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011, http://www.ietf.org/rfc/rfc6234.txt

[SSL3] Netscape, "SSL 3.0 Specification", http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

[VIA2002] Cameron, D., and Regnier, G., "The Virtual Interface Architecture", Intel Press, 2002, ISBN:0971288704.

1.2.2 Informative References

[MC-SMP] Microsoft Corporation, "Session Multiplex Protocol".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-SSCLRT] Microsoft Corporation, "SQL Server Common Language Runtime (CLR) Types Serialization Formats".

[MSDN-Autocommit] Microsoft Corporation, "SQL Server - Autocommit Transactions", http://msdn.microsoft.com/en-us/library/aa386980(VS.85).aspx

[MSDN-BEGIN] Microsoft Corporation, "BEGIN TRANSACTION (Transact SQL)", http://msdn.microsoft.com/en-us/library/ms188929.aspx

[MSDN-BOUND] Microsoft Corporation, "Using Bound Sessions", http://msdn.microsoft.com/en-us/library/ms177480.aspx

[MSDN-BROWSE] Microsoft Corporation, "Browse Mode", http://msdn.microsoft.com/en-us/library/aa936959(SQL.80).aspx

[MSDN-Collation] Microsoft Corporation, "Collation and International Terminology", http://msdn.microsoft.com/en-us/library/ms143726.aspx

[MSDN-ColSets] Microsoft Corporation, "Using Column Sets", http://msdn.microsoft.com/en-us/library/cc280521.aspx

[MSDN-ColSortSty] Microsoft Corporation, "Windows Collation Sorting Style", http://msdn.microsoft.com/en-us/library/ms143515.aspx

[MSDN-COMMIT] Microsoft Corporation, "COMMIT TRANSACTION (Transact SQL)", http://msdn.microsoft.com/en-us/library/ms190295.aspx

[MSDN-DTC] Microsoft Corporation, "Distributed Transaction Coordinator", http://msdn.microsoft.com/en-us/library/ms684146.aspx

[MSDN-INSERT] Microsoft Corporation, "INSERT (Transact-SQL)", http://msdn.microsoft.com/en-us/library/ms174335.aspx

[MSDN-ITrans] Microsoft Corporation, "ITransactionExport::GetTransactionCookie", http://msdn.microsoft.com/en-us/library/ms679869(VS.85).aspx

Copyright © 2014 Microsoft Corporation.

[MSDN-MARS] Microsoft Corporation, "Multiple Active Result Sets (MARS) in SQL Server 2005", http://msdn.microsoft.com/en-us/library/ms345109.aspx

[MSDN-MDAC] Microsoft Corporation, "Microsoft Data Access Components (MDAC) Installation", http://msdn.microsoft.com/en-us/library/ms810805.aspx

[MSDN-NamedPipes] Microsoft Corporation, "Creating a Valid Connection String Using Named Pipes", http://msdn.microsoft.com/en-us/library/ms189307(SQL.100).aspx

[MSDN-QUERYNOTE] Microsoft Corporation, "Using Query Notifications", http://msdn.microsoft.com/en-us/library/ms175110.aspx

[MSDN-SNAC] Microsoft Corporation, "Microsoft SQL Server Native Client and Microsoft SQL Server 2008 Native Client", http://blogs.msdn.com/b/sqlnativeclient/archive/2008/02/27/microsoft-sql-server-native-client-and-microsoft-sql-server-2008-native-client.aspx

[MSDN-SQLCollation] Microsoft Corporation, "Selecting a SQL Collation", http://msdn.microsoft.com/en-us/library/ms144250.aspx

[MSDN-UPDATETEXT] Microsoft Corporation, "UPDATETEXT (Transact-SQL)", http://msdn.microsoft.com/en-us/library/ms189466.aspx

[MSDN-WRITETEXT] Microsoft Corporation, "WRITETEXT (Transact-SQL)", http://msdn.microsoft.com/en-us/library/ms186838.aspx

[NTLM] Microsoft Corporation, "Microsoft NTLM", http://msdn.microsoft.com/en-us/library/aa378749.aspx

[PIPE] Microsoft Corporation, "Named Pipes", http://msdn.microsoft.com/en-us/library/aa365590.aspx

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.txt

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

[SSPI] Microsoft Corporation, "SSPI", http://msdn.microsoft.com/en-us/library/aa380493.aspx

1.3 Overview

The Tabular Data Stream (TDS) Protocol is an application-level protocol used for the transfer of requests and responses between **clients** and database **server** systems. In such systems, the client will typically establish a long-lived connection with the server. Once the connection is established using a transport-level protocol, TDS messages are used to communicate between the client and the server. A database server can also act as the client if needed, in which case a separate TDS connection must be established. Note that the **TDS session** is directly tied to the transport-level session, meaning that a TDS session is established when the transport-level connection is established and the server receives a request to establish a TDS connection. It persists until the transport-level connection is terminated (for example, when a TCP socket is closed). In addition, TDS does not make any assumption about the transport protocol used, but it does assume the transport protocol supports reliable, in-order delivery of the data.

TDS includes facilities for authentication and identification, channel encryption negotiation, issuing of **SQL batches**, stored procedure calls, returning data, and transaction manager requests. Returned data is self-describing and record-oriented. The **data streams** describe the names, types

Copyright © 2014 Microsoft Corporation.

and optional descriptions of the rows being returned. The following diagram depicts a (simplified) typical flow of communication in the TDS Protocol.

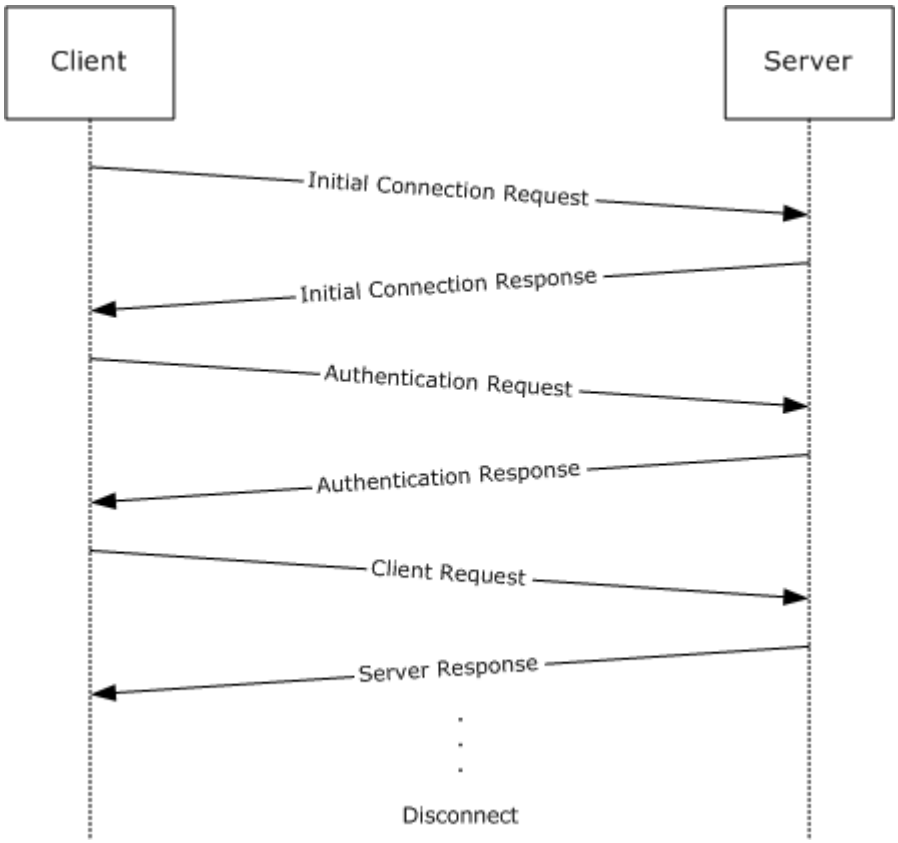

Figure 1: Communication flow in the TDS protocol

The following example is a high-level description of the messages exchanged between the client and the server to execute a simple client request such as the execution of a SQL statement. It is assumed that the client and the server have already established a connection and authentication has succeeded.

```
Client:SQL statement
```

The server executes the **SQL statement** and then sends back the results to the client. The data **columns** being returned are first described by the server (represented as column metadata or COLMETADATA) and then the rows follow. A completion message is sent after all the row data has been transferred.

```
Server:COLMETADATAdata stream
ROWdata stream
.
.
ROWdata stream
DONEdata stream
```


Copyright © 2014 Microsoft Corporation.

For more information about the correlation between data stream and TDS buffer, see section 2.2.4.<1>

Additional details about which Microsoft SQL Server version corresponds to which TDS version number are defined in LOGINACK (section 2.2.7.12).

TDS 7.3.A does not include support for NBCROW and fSparseColumnSet.

1.4 Relationship to Other Protocols

The Tabular Data Stream (TDS) protocol depends upon a network transport connection being established prior to a TDS conversation occurring (the choice of transport protocol is not important to TDS). TDS depends on Transport Layer Security (TLS)/Secure Socket Layer (SSL) for network channel encryption. Although the TDS protocol depends on TLS/SSL to encrypt data transmission, the negotiation of the encryption setting between the client and server and the initial TLS/SSL handshake are handled in the TDS layer.

If the **Multiple Active Result Sets (MARS)** feature [MSDN-MARS] is enabled, then the Session Multiplex Protocol (SMUX) [MC-SMP] is required.

This relationship is illustrated in the following figure.

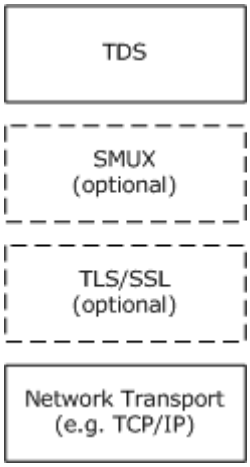

Figure 2: Protocol relationship

1.5 Prerequisites/Preconditions

Throughout this document, it is assumed that the client has already discovered the server and established a network transport connection for use with TDS.

No security association is assumed to have been established at the lower layer before TDS begins functioning. For **Security Support Provider Interface (SSPI)** [SSPI] authentication to be used, SSPI support must be available on both the client and server machines. For channel encryption to be used, TLS/SSL support must be present on both the client and server machines, and a certificate suitable for encryption must be deployed on the server machine. For federated authentication to be used, a library that provides federated authentication support or an equivalent must be present on the server, and the client should be able to generate a token for federated authentication.

Copyright © 2014 Microsoft Corporation.

1.6 Applicability Statement

The TDS protocol is appropriate for use to facilitate request/response communications between an application and a database server in all scenarios where network or local connectivity is available.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas.

- **Supported Transports:** This protocol can be implemented on top of any network transport protocol as discussed in section 2.1.
- **Protocol Versions:** The TDS protocol supports the "TDS 7.0", "TDS 7.1", "TDS 7.2", "TDS 7.3", and "TDS 7.4" explicit dialects. The dialect version is negotiated as part of the LOGIN7 message data stream, which is defined in section 2.2.6.3. (Note that once a protocol feature has been introduced, it is supported in subsequent versions of the TDS protocol until explicitly removed.)
- **Security and Authentication Methods:** The TDS protocol supports **SQL Server User Authentication (SQLAUTH)**. The TDS protocol also supports SSPI authentication and indirectly supports any authentication mechanism that SSPI supports. The use of SSPI in TDS is defined in sections 2.2.6.3 and 3.2.5.1. The TDS protocol also supports federated authentication. The use of federated authentication in TDS is defined in sections 2.2.6.3 and 3.2.5.
- **Localization:** Localization-dependent protocol behavior is specified in sections 2.2.5.1.2 and 2.2.5.6.
- **Capability Negotiation:** This protocol does explicit capability negotiation as specified in this section.

In general, the TDS protocol does not provide facilities for capability negotiation because the complete set of supported features is fixed for each version of the protocol. Certain features such as authentication type are not usually negotiated but rather are requested by the client. However, the protocol supports negotiation for the following two features:

- **Channel encryption:** The encryption behavior that is used for the TDS session is negotiated in the initial messages exchanged by the client and the server.
- **Authentication mechanism for integrated authentication identities:** The authentication mechanism that is used for the TDS session is negotiated in the initial messages exchanged by the client and the server.

For more details about encryption behavior and about how the client and server negotiate between SSPI authentication and federated authentication, see the PRELOGIN description in section 2.2.6.4.

Note that the cipher suite for TLS/SSL and the authentication mechanism for SSPI and federated authentication are negotiated outside the influence of TDS in [RFC2246] and [SSL3].

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

Parameter	TCP port value	Reference
Default SQL Server instance TCP port	1433	[IANAPORT]

Copyright © 2014 Microsoft Corporation.

2 Messages

The formal syntax of all messages is provided in Augmented Backus-Naur Form (ABNF), as specified in [RFC4234].

2.1 Transport

The TDS protocol does not prescribe a specific underlying transport protocol to use on the Internet or on other networks. TDS only presumes a reliable transport that guarantees in-sequence delivery of data.

The chosen transport can be either stream-oriented or message-oriented. If a message-oriented transport is used, any TDS packet sent from a TDS client to a TDS server MUST be contained within a single transport data unit. Any additional mapping of TDS data onto the transport data units of the protocol in question is outside the scope of this specification.

The current version of the TDS protocol has implementations over the following transports:<2>

- TCP [RFC793].
- A reliable transport over the **Virtual Interface Architecture (VIA) interface** [VIA2002].<3>
- Named Pipes (see [PIPE])
- Optionally, the TDS protocol has implementations for the following two protocols on top of the preceding transports:
  - Transport Layer Security (TLS) [RFC2246]/Secure Socket Layer (SSL), in case TLS/SSL encryption is negotiated.
  - **Session Multiplex Protocol (SMUX)** [MC-SMP], in case the Multiple Active Result Sets (MARS) feature [MSDN-MARS] is requested.

2.2 Message Syntax

Character data, such as SQL statements, within a TDS message is in **Unicode**, unless the character data represents the data value of an ASCII data type, such as a non-Unicode data column. Character counts within TDS are a count of characters, rather than bytes, except when explicitly specified as byte counts.

2.2.1 Client Messages

Messages sent from the client to the server are as follows:

- A pre-login record
- A login record
- A SQL batch (in any language that the server will accept)
- A SQL statement followed by its associated binary data (for example, the data for a bulk load SQL statement)
- A remote procedure call
- An attention signal

Copyright © 2014 Microsoft Corporation.

These are briefly described later; detailed descriptions of message contents are in section 2.2.6.

2.2.1.1 Pre-Login

Before a login occurs, a handshake denominated pre-login occurs between client and server, setting up contexts such as encryption and MARS-enabled. For more details, see section 2.2.6.4.

2.2.1.2 Login

When the client makes the determination to establish a TDS protocol connection with the server side, the client sends a login message data stream to the server. The client can have more than one connection to the server, but each one is established separately in the same way. For more details, see section 2.2.6.3.

After the server receives the login record from the client and, if necessary, performs subsequent authentication handshakes (such as when SSPI [SSPI] or federated authentication is used), the server will notify the client that it has either accepted or rejected the connection request. For more details, see section 3.3.5.1.

2.2.1.3 SQL Batch

To send a SQL statement or a batch of SQL statements, the SQL batch, represented by a Unicode string, is copied into the data section of a TDS packet and then sent to the database server that supports SQL. A SQL batch can span more than one TDS packet. For more details, see section 2.2.6.6.

2.2.1.4 Bulk Load

The **bulk insert**/bulk load operation is a case of a SQL statement that consists of a Unicode string followed by binary data. The client sends the **INSERT BULK** SQL statement and then a COLMETADATA token that describes the raw data is sent. Multiple rows of binary data are then sent to the server. The data is not formatted in storage row format but in the format described by the COLMETADATA token. The stream is the same as if the data were being selected from the server rather than being sent to the server. For more details, see section 2.2.6.1.

2.2.1.5 Remote Procedure Call

To execute a remote procedure call (RPC) on the server, the client sends an RPC message data stream to the server. This is a binary stream that contains the RPC name or numeric identifier, options, and parameters. RPCs MUST be in a separate TDS message and not intermixed with SQL statements. There can be several RPCs in one message. For more details, see section 2.2.6.5.

2.2.1.6 Attention

The client can interrupt and cancel the current request by sending an **Attention** message. This is also known as **out-of-band** data, but any TDS packet that is currently being sent MUST be finished before sending the **Attention** message. After the client sends an **Attention** message, the client MUST read until it receives an **Attention** acknowledgment.

If a complete request has been sent to the server, sending a cancel requires sending an **Attention** packet. An example of this behavior is if the client has already sent a request, which has the last packet with EOM bit (0x01) set in status. The **Attention** packet is the only way to interrupt a complete request that has already been sent to the server. For more information, see section 4.19.2.

Copyright © 2014 Microsoft Corporation.

If a complete request has not been sent to the server, the client MUST send the next packet with both ignore bit (0x02) and EOM bit (0x01) set in the status to cancel the request. An example of this behavior is if one or more packets have been sent but the last packet with EOM bit (0x01) set in status has not been sent. Setting the ignore and EOM bit terminates the current request, and the server MUST ignore the current request. When the ignore and EOM bit is set the server will not send an attention acknowledgment but instead return a **table response** with a single DONE token with a status of DONE_ERROR to indicate the incoming request was ignored. For more details about the buffer header status code, see section 2.2.3.1.2.

2.2.1.7 Transaction Manager Request

The client can request that the connection enlist in an [MSDN-DTC] transaction.

2.2.2 Server Messages

Messages sent from the server to the client are:

- A pre-login response
- A login response
- Row data
- The return status of an RPC
- Return parameters of an RPC
- Response completion information
- ERROR and INFO messages
- An attention acknowledgement

These are briefly described below; detailed descriptions of message contents are in section 2.2.6.

2.2.2.1 Pre-Login Response

The pre-login response is a tokenless packet data stream. The data stream consists of the response to the information requested by the client pre-login message. For a detailed description of this stream, see section 2.2.6.4.

2.2.2.2 Login Response

The login response is a token stream consisting of information about the server's characteristics, optional information and error messages, followed by a completion message.

The LOGINACK token data stream includes information about the server interface and the server's product code and name. For a detailed description of the login response data stream, see section 2.2.7.12.

If there are any messages in the login response, an ERROR or INFO token data stream is returned from the server to the client. For more information, see sections 2.2.7.9 and 2.2.7.11.

The server can send, as part of the login response, one or more ENVCHANGE token data streams if the login changed the environment and the associated notification flag was set. An example of an environment change includes the current database context and language setting. For details about the different environment changes, see section 2.2.7.8.

Copyright © 2014 Microsoft Corporation.

A done packet MUST be present as the final part of the login response, and a DONE token data stream is the last thing sent in response to a server login request. For more information about the DONE token data stream, see section 2.2.7.5.

2.2.2.3 Row Data

If the server request results in data being returned, the data will precede any other data streams returned from the server except warnings. Row data MUST be preceded by a description of the column names and data types. For more information about how the column names and data types are described, see section 2.2.7.4.

2.2.2.4 Return Status

When a stored procedure is executed by the server, the server MUST return a status value. This is a 4-byte integer and is sent via the RETURNSTATUS token. A stored procedure execution is requested through either an RPC Batch or a SQL Batch message. For more information, see section 2.2.7.16.

2.2.2.5 Return Parameters

The response format for execution of a stored procedure is identical regardless of whether the request was sent as SQL Batch or RPC Batch. It is always a tabular result-type message.

If the procedure explicitly sends any data, then the message starts with a single token stream of rows, informational messages, and error messages. This data is sent in the usual way.

When the RPC is invoked, some or all of its parameters are designated as output parameters. All output parameters will have values returned from the server. For each output parameter, there is a corresponding return value, sent via the RETURNVALUE token. The RETURNVALUE token data stream is also used for sending back the value returned by a user-defined function (UDF), if it is called as an RPC. For more details about the RETURNVALUE token, see section 2.2.7.17.

2.2.2.6 Response Completion ("DONE")

The client reads results in logical units and can tell when all results have been received by examining the DONE token data stream.

When executing a batch of SQL statements, the server MUST return a DONE token data stream for each set of results. All but the last DONE will have the DONE_MORE bit set in the **Status** field of the DONE token data stream. Therefore, the client can always tell after reading a DONE whether or not there are more results. For more details about the DONE token, see section 2.2.7.5.

For stored procedures, completion of SQL statements in the stored procedure is indicated by a DONEINPROC token data stream for each SQL statement and a DONEPROC token data stream for each completed stored procedure. For more details about DONEINPROC and DONEPROC tokens, see section 2.2.7.6 and 2.2.7.7, respectively.

2.2.2.7 ERROR and INFO Messages

Besides returning description of Row data and the data itself, TDS provides a token data stream type for the server to send error or informational messages to the client. These are the INFO token data stream and the ERROR token data stream.

Copyright © 2014 Microsoft Corporation.

2.2.2.8 Attention Acknowledgment

After a client has sent an interrupt signal to the server, the client MUST read returning data until the interrupt has been acknowledged. Attentions are acknowledged in the DONE token data stream.

2.2.3 Packets

A packet is the unit written or read at one time. A message can consist of one or more packets. A packet always includes a packet header and is usually followed by packet data that contains the message. Each new message starts in a new packet.

In practice, both the client and server will try to read a packet full of data. They will pick out the header to see how much more (or less) data there is in the communication.

At login time, clients MAY specify a requested "packet" size as part of the LOGIN7 message stream. This identifies the size used to break large messages into different "packets". Server acknowledgment of changes in the negotiated packet size is transmitted back to the client via ENVCHANGE token stream. The negotiated packet size is the maximum value that can be specified in the **Length** packet header field described in section 2.2.3.1.3.

Starting with TDS 7.3, the following behavior MUST also be enforced. For requests sent to the server larger than the current negotiated "packet" size, the client MUST send all but the last packet with a total number of bytes equal to the negotiated size. Only the last packet in the request can contain an actual number of bytes smaller than the negotiated packet size. If any of the preceding packets are sent with a length less than the negotiated packet size, the server SHOULD disconnect the client when the next network payload arrives.

2.2.3.1 Packet Header

To implement messages on top of existing, arbitrary transport layers, a packet header is included as part of the packet. The packet header precedes all data within the packet. It is always 8 bytes in length. Most importantly, the buffer header states the **Type** and **Length** of the entire packet.

The following is a detailed description of each item within the packet header.

2.2.3.1.1 Type

Type defines the type of message. **Type** is a 1-byte unsigned char. Types are as follows:

Value	Description	Buffer data?
1	SQL batch. This can be any language that the server understands.	Yes
2	Pre-TDS7 login (only used by legacy clients older than Microsoft SQL Server 7.0).	Yes
3	RPC.	Yes
4	Tabular result. This indicates a stream that contains the server response to a client request.	Yes
5	Unused.	-
6	Attention signal.	No
7	Bulk load data. This type is used to send binary data to the server.	Yes

Copyright © 2014 Microsoft Corporation.

Value	Description	Buffer data?
8-13	Unused.	-
14	Transaction manager request.	Yes
15	Unused.	-
16	TDS7 login (MUST be used by all clients that support SQL Server 7.0 or later).	Yes
17	SSPI message.	Yes
18	Pre-login message.	Yes

If an unknown **Type** is specified, the message receiver SHOULD disconnect the connection. If a valid **Type** is specified, but is unexpected (per section 3), the message receiver SHOULD disconnect the connection. This applies to both the client and the server. For example, the server could disconnect the connection if the server receives a message with **Type** equal 16 when the connection is already logged in.

The following table highlights which messages, as described previously in sections 2.2.1 and 2.2.2, correspond to which packet header type.

Message type	Client or server message	Buffer header type
Pre-Login	Client	2 or 18 depending on whether the client supports TDS v7.0+
Login	Client	16 + 17 (if Integrated authentication)
SQL batch	Client	1
Bulk load	Client	7
RPC	Client	3
Attention	Client	6
Transaction Manager Request	Client	14
FeatureExtAck	Server	4
Pre-Login Response	Server	4
Login Response	Server	4
Row Data	Server	4
Return Status	Server	4
Return Parameters	Server	4
Response Completion ("DONE")	Server	4
Session State	Server	4

Copyright © 2014 Microsoft Corporation.

Message type	Client or server message	Buffer header type
Error and Info Messages	Server	4
Attention Acknowledgement	Server	4

2.2.3.1.2 Status

Status is a bit field used to indicate the message state. **Status** is a 1-byte unsigned char. The following **Status** bit flags are defined.

Value	Description
0x00	"Normal" message.
0x01	End of message (EOM). The packet is the last packet in the whole request.
0x02	(From client to server) Ignore this event (0x01 MUST also be set).
0x08	RESETCONNECTION (Introduced in TDS 7.1) (From client to server) Reset this connection before processing event. Only set for event types Batch, RPC, or **Distributed Transaction Coordinator (DTC)** Request. If clients want to set this bit, it MUST be part of the first packet of the message. This signals the server to clean up the environment state of the connection back to the default environment setting, effectively simulating a logout and a subsequent login, and provides server support for connection pooling. This bit SHOULD be ignored if it is set in a packet that is not the first packet of the message. This status bit MUST NOT be set in conjunction with the RESETCONNECTIONSKIPTRAN bit. Distributed transactions and isolation levels will not be reset.
0x10	RESETCONNECTIONSKIPTRAN (Introduced in TDS 7.3) (From client to server) Reset the connection before processing event but do not modify the transaction state (the state will remain the same before and after the reset). The transaction in the session can be a local transaction that is started from the session or it can be a distributed transaction in which the session is enlisted. This status bit MUST NOT be set in conjunction with the RESETCONNECTION bit. Otherwise identical to RESETCONNECTION.

All other bits are not used and MUST be ignored.

2.2.3.1.3 Length

Length is the size of the packet including the 8 bytes in the packet header. It is the number of bytes from the start of this header to the start of the next packet header. Length is a 2-byte, unsigned short int and is represented in network byte order (**big-endian**). Starting with TDS 7.3, the Length MUST be the negotiated packet size when sending a packet from client to server, unless it is the last packet of a request (that is, the EOM bit in Status is ON), or the client has not logged in.

2.2.3.1.4 SPID

Spid is the process ID on the server, corresponding to the current connection. This information is sent by the server to the client and is useful for identifying which thread on the server sent the TDS packet. It is provided for debugging purposes. The client MAY send the SPID value to the server. If

Copyright © 2014 Microsoft Corporation.

the client does not, then a value of 0x0000 SHOULD be sent to the server. This is a 2-byte value and is represented in network byte order (big-endian).

2.2.3.1.5 PacketID

PacketID is used for numbering message packets that contain data in addition to the packet header. PacketID is a 1-byte, unsigned char. Each time packet data is sent, the value of **PacketID** is incremented by 1, modulo 256. This allows the receiver to track the sequence of TDS packets for a given message. This value is currently ignored.

2.2.3.1.6 Window

This 1 byte is currently not used. This byte SHOULD be set to 0x00 and SHOULD be ignored by the receiver.

2.2.3.2 Packet Data

Packet data for a given message follows the packet header (see **Type** in section 2.2.3.1.1 for messages that contain packet data). As previously stated, a message can span more than one packet. Because each new message MUST always begin within a new packet, a message that spans more than one packet only occurs if the data to be sent exceeds the maximum packet data size, which is computed as (negotiated packet size - 8 bytes), where the 8 bytes represents the size of the packet header.

If a stream spans more than one packet, then the EOM bit of the packet header Status code MUST be set to 0 for every packet header. The EOM bit MUST be set to 1 in the last packet to signal that the stream ends. In addition, the **PacketID** field of subsequent packets MUST be incremented as defined in section 2.2.3.1.5.

2.2.4 Packet Data Token and Tokenless Data Streams

The messages contained in packet data that pass between the client and the server can be one of two types: a "token stream" or a "tokenless stream". A token stream consists of one or more "tokens" each followed by some token-specific data. A "token" is a single byte identifier used to describe the data that follows it (for example contains token data type, token data length, and so on). Tokenless streams are typically used for simple messages. Messages that might require a more detailed description of the data within it are sent as a token stream. The following table highlights which messages, as described previously in sections 2.2.1 and 2.2.2, use token streams and which do not.

Message type	Client or server message	Token stream?
Pre-Login	Client	No
Login	Client	No
SQL Command	Client	No
SQL Command with Binary Data	Client	Yes
Remote Procedure Call (RPC)	Client	Yes
Attention	Client	No
Transaction Manager Request	Client	No

Copyright © 2014 Microsoft Corporation.

Message type	Client or server message	Token stream?
Pre-Login Response	Server	No
FeatureExtAck	Server	Yes
Login Response	Server	Yes
Row Data	Server	Yes
Return Status	Server	Yes
Return Parameters	Server	Yes
Response Completion ("DONE")	Server	Yes
Session State	Server	Yes
Error and Info Messages	Server	Yes
Attention Acknowledgement	Server	No

2.2.4.1 Tokenless Stream

As shown in the previous section, some messages do not use tokens to describe the data portion of the data stream. In these cases, all the information required to describe the packet data is contained in the packet header. This is referred to as a tokenless stream and is essentially just a collection of packets and data.

2.2.4.2 Token Stream

More complex messages (for example, colmetadata, row data, and data type data) are constructed by using tokens. As previously described, a token stream consists of a single byte identifier, followed by token-specific data. The definition of different token stream can be found in section 2.2.7.

2.2.4.2.1 Token Definition

There are four classes of token definitions:

- Zero Length Token(xx01xxxx)
- Fixed Length Token(xx11xxxx)
- Variable Length Tokens(xx10xxxx)
- Variable Count Tokens(xx00xxxx)

The following sections specify the bit pattern of each token class, various extensions to this bit pattern for a given token class, and a description of its function(s).

2.2.4.2.1.1 Zero Length Token(xx01xxxx)

This class of token is not followed by a length specification. There is no data associated with the token. A zero length token always has the following bit sequence:

Copyright © 2014 Microsoft Corporation.

0	1	2	3	4	5	6	7
x	x	0	1	x	x	x	x

In the diagram above, x denotes a bit position that can contain the bit value 0 or 1.

2.2.4.2.1.2 Fixed Length Token(xx11xxxx)

This class of token is followed by 1, 2, 4, or 8 bytes of data. No length specification follows this token because the length of its associated data is encoded in the token itself. The different fixed data-length token definitions take the form of one of the following bit sequences, depending on whether the token is followed by 1, 2, 4, or 8 bytes of data.

0	1	2	3	4	5	6	7	Description
x	x	1	1	0	0	x	x	Token is followed by 1 byte of data.
x	x	1	1	0	1	x	x	Token is followed by 2 bytes of data.
x	x	1	1	1	0	x	x	Token is followed by 4 bytes of data.
x	x	1	1	1	1	x	x	Token is followed by 8 bytes of data.

In the diagram above, x denotes a bit position that can contain the bit value 0 or 1.

Fixed-length tokens are used by the following data types: *bigint, int*, *smallint*, *tinyint, float, real, money, smallmoney, datetime, smalldatetime,* and *bit*. The type definition is always represented in COLMETADATA and ALTMETADATA data streams as a single byte Type. Additional details are specified in section 2.2.5.3.1.

2.2.4.2.1.3 Variable Length Tokens(xx10xxxx)

This class of token definition is followed by a length specification. The length (in bytes) of this length is included in the token itself as a Length value (see section 2.2.7.3 for information about the Length rule of the COLINFO token stream).

There are two data types that are of variable length. These are real variable length data types like char and binary and nullable data types, which are either their normal fixed length corresponding to their type_info, or a special length if null.

Char and binary data types have values that are either null or 0 to 65534 (0x0000 to 0xFFFE) bytes in length. Null is represented by a length of 65535 (0xFFFF). A char or binary, which cannot be null, can still have a length of zero (for example an empty value). A program that MUST pad a value to a fixed length will typically add blanks to the end of a char and binary zeros to the end of a binary.

Text and image data types have values that are either null, or 0 to 2 gigabytes (0x00000000 to 0x7FFFFFFF bytes) in length. Null is represented by a length of -1 (0xFFFFFFFF). No other length specification is supported.

Other nullable data types have a length of 0 if they are null.

Copyright © 2014 Microsoft Corporation.

2.2.4.2.1.4 Variable Count Tokens(xx00xxxx)

This class of token definition is followed by a count of the number of fields that follow the token. Each field length is dependent on the token type. The total length of the token can be determined only by walking the fields. A variable count token always has its third and fourth bits set to 0.

0	1	2	3	4	5	6	7
x	x	0	0	x	x	x	x

In the diagram above, x denotes a bit position that can contain the bit value 0 or 1.

Currently there are two variable count tokens. COLMETADATA and ALTMETADATA both use a 2-byte count.

2.2.4.3 Done and Attention Tokens

The DONE token marks the end of the response for each executed SQL statement. Based on the SQL statement and the context in which it is executed, the server MAY generate a DONEPROC or DONEINPROC token instead.

The attention signal is sent by using the out-of-band write provided by the network library. An out-of-band write is the ability to send the attention signal no matter if the sender is in the middle of sending or processing a message or simply sitting idle. If that function is not supported, the client MUST simply read and discard all of the data, except SESSIONSTATE data, from the server until the final DONE token, which acknowledges that the attention signal is read.<4>

2.2.5 Grammar Definition for Token Description

The Tabular Data Stream consists of a variety of messages. Each message consists of a set of bytes transmitted in a predefined order. This predefined order or grammar can be specified using Augmented Backus-Naur Form [RFC4234]. Details can be found in the following subsections.

2.2.5.1 General Rules

Data structure encodings in TDS are defined in terms of the following fundamental definitions:

BIT: A single bit value of either 0 or 1.

```
BIT = %b0 / %b1
```

BYTE: An unsigned single byte (8-bit) value. The range is 0 to 255.

```
BYTE = 8BIT
```

BYTELEN: An unsigned single byte (8-bit) value representing the length of the associated data. The range is 0 to 255.

```
BYTELEN = BYTE
```


Copyright © 2014 Microsoft Corporation.

USHORT: An unsigned 2-byte (16-bit) value. The range is 0 to 65535.

```
USHORT = 2BYTE
```

LONG: A signed 4-byte (32-bit) value. The range is -(2^31) to (2^31)-1.

```
LONG = 4BYTE
```

ULONG: An unsigned 4-byte (32-bit) value. The range is 0 to (2^32)-1

```
ULONG = 4BYTE
```

DWORD: An unsigned 4-byte (32-bit) value. The range when used as a numeric value is 0 to (2^32)-1.

```
DWORD = 32BIT
```

LONGLONG: A signed 8-byte (64-bit) value. The range is –(2^63) to (2^63)-1.

```
LONGLONG = 8BYTE
```

ULONGLONG: An unsigned 8-byte (64-bit) value. The range is 0 to (2^64)-1.

```
ULONGLONG = 8BYTE
```

UCHAR: An unsigned single byte (8-bit) value representing a character. The range is 0 to 255.

```
UCHAR = BYTE
```

USHORTLEN: An unsigned 2-byte (16-bit) value representing the length of the associated data. The range is 0 to 65535.

```
USHORTLEN = 2BYTE
```

USHORTCHARBINLEN: An unsigned 2-byte (16-bit) value representing the length of the associated character or binary data. The range is 0 to 8000.

```
USHORTCHARBINLEN = 2BYTE
```


Copyright © 2014 Microsoft Corporation.

LONGLEN: A signed 4-byte (32-bit) value representing the length of the associated data. The range is -(2^31) to (2^31)-1.

```
LONGLEN = 4BYTE
```

ULONGLONGLEN: An unsigned 8-byte (64-bit) value representing the length of the associated data. The range is 0 to (2^64)-1.

```
ULONGLONGLEN = 8BYTE
```

PRECISION: An unsigned single byte (8-bit) value representing the precision of a numeric number.

```
PRECISION = 8BIT
```

SCALE: An unsigned single byte (8-bit) value representing the scale of a numeric number.

```
SCALE = 8BIT
```

GEN_NULL: A single byte (8-bit) value representing a NULL value.

```
GEN_NULL = %x00
```

CHARBIN_NULL: A 2-byte (16-bit) or 4-byte (32-bit) value representing a T-SQL NULL value for a character or binary data type. Please refer to TYPE_VARBYTE (see section 2.2.5.2.3) for additional details.

```
CHARBIN_NULL = (%xFF %xFF) / (%xFF %xFF %xFF %xFF)
```

FRESERVEDBIT: A FRESERVEDBIT is a BIT value used for padding that does not transmit information. FRESERVEDBIT fields SHOULD be set to %b0 and MUST be ignored on receipt.

```
FRESERVEDBIT = %b0
```

FRESERVEDBYTE: A FRESERVEDBYTE is a BYTE value used for padding that does not transmit information. FRESERVEDBYTE fields SHOULD be set to %x00 and MUST be ignored on receipt.

```
FRESERVEDBYTE = %x00
```

UNICODECHAR: A single Unicode character in UCS-2 encoding, as specified in UNICODE [UNICODE].

Copyright © 2014 Microsoft Corporation.

```
UNICODECHAR = 2BYTE
```

Notes

- All integer types are represented in reverse byte order (**little-endian**) unless otherwise specified.
- FRESERVEDBIT and FRESERVEDBYTE are often used to pad unused parts of a byte or bytes. The value of these reserved bits SHOULD be ignored. These elements are generally set to 0.

2.2.5.1.1 Least Significant Bit Order

Certain tokens will possess rules that comprise an array of independent bits. These are typically "flag" rules in which each bit is a flag indicating that a specific feature or option is enabled/requested. Normally, the bit array will be arranged in least significant bit order (or typical array index order) meaning that the first listed flag is placed in the least significant bit position (identifying the least significant bit as one would in an integer variable). For example, if *Fn* is the *nth* flag, then the following rule definition:

```
FLAGRULE = F0 F1 F2 F3 F4 F5 F6 F7
```

would be observed on the wire in the natural value order F7F6F5F4F3F2F1F0.

If the rule contains 16 bits, then the order of the bits observed on the wire will follow the little-endian byte ordering. For example, the following rule definition:

```
FLAGRULE = F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
```

will have the following order on the wire: F7F6F5F4F3F2F1F0 F15F14F13F12F11F10F9F8.

2.2.5.1.2 Collation Rule Definition

The collation rule is used to specify collation information for character data or metadata describing character data. This is typically specified as part of the LOGIN7 message or part of a column definition in server results containing character data. For more information about column definition, see COLMETADATA.

```
LCID            =   20BIT

fIgnoreCase     =   BIT
fIgnoreAccent   =   BIT
fIgnoreWidth    =   BIT
fIgnoreKana     =   BIT
fBinary         =   BIT
fBinary2        =   BIT
ColFlags        =   fIgnoreCase fIgnoreAccent fIgnoreKana
                    fIgnoreWidth fBinary fBinary2 FRESERVEDBIT
                    FRESERVEDBIT
Version         =   4BIT
SortId          =   BYTE
```


Copyright © 2014 Microsoft Corporation.

```
COLLATION        =   LCID ColFlags Version SortId
```

A SQL (SortId==1) collation is one of a predefined set of sort orders. It is identified by having SortId with values as described by [MSDN-SQLCollation].

For a SortId==0 collation, the LCID bits correspond to a LocaleId as defined by the National Language Support (NLS) functions. For more details, see [MS-LCID].

Notes<5>

The sorting styles used in SQL Server are described at [MSDN-ColSortSty].

- If COLLATION specifies 0x00 00 00 00 00, then it indicates request for use of raw collation.
- ColFlags is represented in least significant bit order.

2.2.5.2 Data Stream Types

2.2.5.2.1 Unknown Length Data Streams

Unknown length data streams can be used by tokenless data streams. It is a stream of bytes. The number of bytes within the data stream is defined in the packet header as specified in section 2.2.3.1.

```
BYTESTREAM       =   *BYTE
UNICODESTREAM    =   *(2BYTE)
```

2.2.5.2.2 Variable-Length Data Streams

Variable-length data streams consist of a stream of characters or a stream of bytes. The two types are similar, in that they both have a length rule and a data rule.

Characters

Variable-length character streams are defined by a length field followed by the data itself. There are two types of variable-length character streams, each dependent on the size of the length field (for example, a BYTE or USHORT). If the length field is zero, then no data follows the length field.

```
B_VARCHAR        =   BYTELEN *CHAR
US_VARCHAR       =   USHORTLEN *CHAR
```

Note that the lengths of B_VARCHAR and US_VARCHAR are given in Unicode characters.

Generic Bytes

Similar to the variable-length character stream, variable-length byte streams are defined by a length field followed by the data itself. There are three types of variable-length byte streams, each dependent on the size of the length field (for example, a BYTE, USHORT, or LONG). If the value of the length field is zero, then no data follows the length field.

```
B_VARBYTE        =   BYTELEN *BYTE
US_VARBYTE       =   USHORTLEN *BYTE
```


Copyright © 2014 Microsoft Corporation.

```
L_VARBYTE        =    LONGLEN *BYTE
```

2.2.5.2.3 Data Type Dependent Data Streams

Some messages contain variable data types. The actual type of a given variable data type is dependent on the type of the data being sent within the message as defined in the TYPE_INFO rule.

For example, the RPCRequest message contains the TYPE_INFO and TYPE_VARBYTE rules. These two rules contain data of a type that is dependent on the actual type used in the value of the FIXEDLENTYPE or VARLENTYPE rules of the TYPE_INFO rule.

Data type-dependent data streams occur in three forms: integers, fixed and variable bytes, and partially length-prefixed bytes.

Integers

Data type-dependent integers can be either a BYTELEN, USHORTCHARBINLEN, or LONGLEN in length. This length is dependent on the TYPE_INFO associated with the message. If the data type (for example, FIXEDLENTYPE or VARLENTYPE rule of the TYPE_INFO rule) is of type SSVARIANTTYPE, TEXTTYPE, NTEXTTYPE, or IMAGETYPE, the integer length is LONGLEN. If the data type is BIGCHARTYPE, BIGVARCHARTYPE, NCHARTYPE, NVARCHARTYPE, BIGBINARYTYPE, or BIGVARBINARYTYPE, the integer length is USHORTCHARBINLEN. For all other data types, the integer length is BYTELEN.

```
TYPE_VARLEN      =    BYTELEN
                      /
                      USHORTCHARBINLEN
                      /
                      LONGLEN
```

Fixed and Variable Bytes

The data type to be used in a data type-dependent byte stream is defined by the TYPE_INFO rule associated with the message.

For variable-length types, with the exception of PLP (see Partially Length-prefixed Bytes below), the TYPE_VARLEN value defines the length of the data to follow. As described above, the TYPE_INFO rule defines the type of TYPE_VARLEN (for example BYTELEN, USHORTCHARBINLEN, or LONGLEN).

For fixed-length types, the TYPE_VARLEN rule is not present. In these cases the number of bytes to be read is determined by the TYPE_INFO rule (for example, if "INT2TYPE" is specified as the value for the FIXEDLENTYPE rule of the TYPE_INFO rule, 2 bytes should be read, as "INT2TYPE" is always 2 bytes in length. For more details, see Data Types Definitions).

The data following this can be a stream of bytes or a NULL value. The 2-byte CHARBIN_NULL rule is used for BIGCHARTYPE, BIGVARCHARTYPE, NCHARTYPE, NVARCHARTYPE, BIGBINARYTYPE, and BIGVARBINARYTYPE types, and the 4-byte CHARBIN_NULL rule is used for TEXTTYPE, NTEXTTYPE, and IMAGETYPE. The GEN_NULL rule applies to all other types aside from PLP:

```
TYPE_VARBYTE = GEN_NULL / CHARBIN_NULL / PLP_BODY
               / ([TYPE_VARLEN] *BYTE)
```


Copyright © 2014 Microsoft Corporation.

Partially Length-prefixed Bytes

Unlike fixed or variable byte stream formats, Partially length-prefixed bytes (PARTLENTYPE), introduced in TDS 7.2, do not require the full data length to be specified before the actual data is streamed out. Thus, it is ideal for those applications where the data length is not known upfront (that is, xml serialization). A value sent as PLP can be either NULL, a length followed by chunks (as defined by PLP_CHUNK), or an unknown length token followed by chunks, which MUST end with a PLP_TERMINATOR. The rule below describes the stream format (for example, the format of a singleton PLP value):

```
PLP_BODY=     PLP_NULL
                      /
                      ((ULONGLONGLEN / UNKNOWN_PLP_LEN)
                      *PLP_CHUNK PLP_TERMINATOR)

PLP_NULL          =   %xFFFFFFFFFFFFFFFF

UNKNOWN_PLP_LEN   =   %xFFFFFFFFFFFFFFFE

PLP_CHUNK         =   ULONGLEN 1*BYTE

PLP_TERMINATOR    =   %x00000000
```

Notes

- TYPE_INFO rule specifies a Partially Length-prefixed Data type (PARTLENTYPE, see 2.2.5.4.3).
- In the UNKNOWN_PLP_LEN case, the data is represented as a series of zero or more chunks, each consisting of the length field followed by length bytes of data (see the PLP_CHUNK rule). The data is terminated by PLP_TERMINATOR (which is essentially a zero-length chunk).
- In the actual data length case, the ULONGLONGLEN specifies the length of the data and is followed by any number of PLP_CHUNKs containing the data. The length of the data specified by ULONGLONGLEN is used as a hint for the receiver. The receiver SHOULD validate that the length value specified by ULONGLONGLEN matches the actual data length.

2.2.5.3 Packet Data Stream Headers - ALL_HEADERS Rule Definition

Message streams can be preceded by a variable number of headers as specified by the ALL_HEADERS rule. The ALL_HEADERS rule, the Query Notifications header, and the Transaction Descriptor header were introduced in TDS 7.2. The Trace Activity header was introduced in TDS 7.4.

The list of headers that are applicable to the different types of messages are described in the following table.

Stream headers MUST be present only in the first packet of requests that span more than one packet. The ALL_HEADERS rule applies only to the three client request types defined in the table below and MUST NOT be included for other request types. For the applicable request types, each header MUST appear at most once in the stream or packet's ALL_HEADERS field.

Header	Value	SQLBatch	RPCRequest	TransactionManagerRequest
Query Notifications	0x00 01	Optional	Optional	Disallowed

Copyright © 2014 Microsoft Corporation.

Header	Value	SQLBatch	RPCRequest	TransactionManagerRequest
Transaction Descriptor	0x00 02	Required	Required	Required
Trace Activity	0x00 03	Optional	Optional	Optional

Stream-Specific Rules:

```
TotalLength      =   DWORD    ;including itself
HeaderLength     =   DWORD    ;including itself
HeaderType       =   USHORT;
HeaderData       =   *BYTE
Header           =   HeaderLength HeaderType HeaderData
```

Stream Definition:

```
ALL_HEADERS      =   TotalLength 1*Header
```

Parameter	Description
TotalLength	Total length of ALL_HEADERS stream.
HeaderLength	Total length of an individual header.
HeaderType	The type of header, as defined by the value field in the preceding table.
HeaderData	The data stream for the header. See header definitions in the following subsections.
Header	A structure containing a single header.

2.2.5.3.1 Query Notifications Header

This packet data stream header allows the client to specify that a notification is desired on the results of the request. The contents of the header specify the information necessary for delivery of the notification. For more information about **query notification** functionality for a database server that supports SQL, see [MSDN-QUERYNOTE].

Stream Specific Rules:

```
NotifyId         =   USHORT UNICODESTREAM ; user specified value
                                            when subscribing to the
                                            query notification
SSBDeployment    =   USHORT UNICODESTREAM ;
NotifyTimeout    =   ULONG                ; duration in which the query
                                            notification subscription
                                            is valid
```

The USHORT field defined within the NotifyId and SSBDeployment rules specifies the length, in bytes, of the actual data value, defined by the UNICODESTREAM, that follows it.

Stream Definition:

Copyright © 2014 Microsoft Corporation.

```
Header Data      =   NotifyId
                     SSBDeployment
                     [NotifyTimeout]
```

2.2.5.3.2 Transaction Descriptor Header

This packet data stream contains information regarding transaction descriptor and number of outstanding requests as they apply to [MSDN-MARS].

The TransactionDescriptor MUST be 0, and OutstandingRequestCount MUST be 1 if the connection is operating in AutoCommit mode. For more information about autocommit transactions, see [MSDN-Autocommit].

Stream-Specific Rules:

```
OutstandingRequestCount =   DWORD     ; number of requests currently
                                        active on the connection
TransactionDescriptor   =   ULONGLONG ; For each connection, a number that
                                        uniquely
                                        identifies the transaction
                                        the request is associated
                                        with.
                                        Initially generated by
                                        the server when a new transaction
                                        is created and returned to
                                        the client as part of the
                                        ENVCHANGE token stream.
```

For more information about processing the Transaction Descriptor header, see section 2.2.6.8.

Stream Definition:

```
Header Data      =   TransactionDescriptor
                     OutstandingRequestCount
```

2.2.5.3.3 Trace Activity Header

This packet data stream contains a client trace activity ID intended to be used by the server for debugging purposes, to allow correlating the server's processing of the request with the client request.

A client MUST NOT send a Trace Activity Header when the negotiated TDS major version is less than 7.4. If the negotiated TDS major version is less than TDS 7.4 and the server receives a Trace Activity Header token, the server MUST reject the request with a TDS protocol error.

Stream-Specific Rules:

```
ActivityId = 20BYTE                 ; client Activity ID
                                    ; for debugging purposes
```

Stream Definition:

Copyright © 2014 Microsoft Corporation.

```
Header Data      =   ActivityId
```

2.2.5.4 Data Type Definitions

The subsections within this section describe the different sets of data types and how they are categorized. Specifically, data values are interpreted and represented in association with their data type. Details about each data type categorization are described in the following sections.

2.2.5.4.1 Fixed-Length Data Types

The fixed-length data types include the following types.

```
NULLTYPE         =   %x1F  ; Null
INT1TYPE         =   %x30  ; TinyInt
BITTYPE          =   %x32  ; Bit
INT2TYPE         =   %x34  ; SmallInt
INT4TYPE         =   %x38  ; Int
DATETIM4TYPE     =   %x3A  ; SmallDateTime
FLT4TYPE         =   %x3B  ; Real
MONEYTYPE        =   %x3C  ; Money
DATETIMETYPE     =   %x3D  ; DateTime
FLT8TYPE         =   %x3E  ; Float
MONEY4TYPE       =   %x7A  ; SmallMoney
INT8TYPE         =   %x7F  ; BigInt

FIXEDLENTYPE     =   NULLTYPE
                     /
                     INT1TYPE
                     /
                     BITTYPE
                     /
                     INT2TYPE
                     /
                     INT4TYPE
                     /
                     DATETIM4TYPE
                     /
                     FLT4TYPE
                     /
                     MONEYTYPE
                     /
                     DATETIMETYPE
                     /
                     FLT8TYPE
                     /
                     MONEY4TYPE
                     /
                     INT8TYPE
```

Non-nullable values are returned using these fixed-length data types. There is no data associated with NULLTYPE.<6> For the rest of the fixed-length data types, the length of data is predefined by the type. There is no TYPE_VARLEN field in the TYPE_INFO rule for these types. In the TYPE_VARBYTE rule for these types, the TYPE_VARLEN field is BYTELEN, and the value is 1 for INT1TYPE/BITTYPE, 2 for INT2TYPE, 4 for INT4TYPE/DATETIM4TYPE/FLT4TYPE/MONEY4TYPE, and 8 for MONEYTYPE/DATETIMETYPE/FLT8TYPE/INT8TYPE. The value represents the number of bytes of

Copyright © 2014 Microsoft Corporation.

data to be followed. The SQL data types of the corresponding fixed-length data types are in the comment part of each data type.

2.2.5.4.2 Variable-Length Data Types

The data type token values defined in this section have a length value associated with the data type because the data values corresponding to these data types are represented by a variable number of bytes.

```
GUIDTYPE            =   %x24  ; UniqueIdentifier
INTNTYPE            =   %x26  ; (see below)
DECIMALTYPE         =   %x37  ; Decimal (legacy support)
NUMERICTYPE         =   %x3F  ; Numeric (legacy support)
BITNTYPE            =   %x68  ; (see below)
DECIMALNTYPE        =   %x6A  ; Decimal
NUMERICNTYPE        =   %x6C  ; Numeric
FLTNTYPE            =   %x6D  ; (see below)
MONEYNTYPE          =   %x6E  ; (see below)
DATETIMNTYPE        =   %x6F  ; (see below)
DATENTYPE           =   %x28  ; (introduced in TDS 7.3)
TIMENTYPE           =   %x29  ; (introduced in TDS 7.3)
DATETIME2NTYPE      =   %x2A  ; (introduced in TDS 7.3)
DATETIMEOFFSETNTYPE =   %x2B  ; (introduced in TDS 7.3)
CHARTYPE            =   %x2F  ; Char (legacy support)
VARCHARTYPE         =   %x27  ; VarChar (legacy support)
BINARYTYPE          =   %x2D  ; Binary (legacy support)
VARBINARYTYPE       =   %x25  ; VarBinary (legacy support)

BIGVARBINTYPE       =   %xA5  ; VarBinary
BIGVARCHRTYPE       =   %xA7  ; VarChar
BIGBINARYTYPE       =   %xAD  ; Binary
BIGCHARTYPE         =   %xAF  ; Char
NVARCHARTYPE        =   %xE7  ; NVarChar
NCHARTYPE           =   %xEF  ; NChar
XMLTYPE             =   %xF1  ; XML (introduced in TDS 7.2)
UDTTYPE             =   %xF0  ; CLR-UDT (introduced in TDS 7.2)

TEXTTYPE            =   %x23  ; Text
IMAGETYPE           =   %x22  ; Image
NTEXTTYPE           =   %x63  ; NText
SSVARIANTTYPE       =   %x62  ; Sql_Variant (introduced in TDS 7.2)

BYTELEN_TYPE        =   GUIDTYPE
                        /
                        INTNTYPE
                        /
                        DECIMALTYPE
                        /
                        NUMERICTYPE
                        /
                        BITNTYPE
                        /
                        DECIMALNTYPE
                        /
                        NUMERICNTYPE
                        /
                        FLTNTYPE
                        /
```


Copyright © 2014 Microsoft Corporation.

```
MONEYNTYPE
/
DATETIMNTYPE
/
DATENTYPE
/
TIMENTYPE
/
DATETIME2NTYPE
/
DATETIMEOFFSETNTYPE
/
CHARTYPE
/
VARCHARTYPE
/
BINARYTYPE
/
VARBINARYTYPE  ; the length value associated
                 with these data types is
                 specified within a BYTE
```

For MONEYNTYPE, the only valid lengths are 0x04 and 0x08, which map to smallmoney and money SQL data types respectively.

For DATETIMNTYPE, the only valid lengths are 0x04 and 0x08, which map to smalldatetime and datetime SQL data types respectively.

For INTNTYPE, the only valid lengths are 0x01, 0x02, 0x04, and 0x08, which map to tinyint, smallint, int, and bigint SQL data types respectively.

For FLTNTYPE, the only valid lengths are 0x04 and 0x08, which map to 7-digit precision float and 15-digit precision float SQL data types respectively.

For GUIDTYPE, the only valid lengths are 0x10 for non-null instances and 0x00 for NULL instances.

For BITNTYPE, the only valid lengths are 0x01 for non-null instances and 0x00 for NULL instances.

For DATENTYPE, the only valid lengths are 0x03 for non-NULL instances and 0x00 for NULL instances.

For TIMENTYPE, the only valid lengths (along with the associated scale value) are:

SCALE	**1**	**2**	**3**	**4**	**5**	**6**	**7**
LENGTH	0x03	0x03	0x04	0x04	0x05	0x05	0x05

For DATETIME2NTYPE, the only valid lengths (along with the associated scale value) are:

SCALE	**1**	**2**	**3**	**4**	**5**	**6**	**7**
LENGTH	0x06	0x06	0x07	0x07	0x08	0x08	0x08

For DATETIMEOFFSETNTYPE, the only valid lengths (along with the associated scale value) are:

Copyright © 2014 Microsoft Corporation.

SCALE	1	2	3	4	5	6	7
LENGTH	0x08	0x08	0x09	0x09	0x0A	0x0A	0x0A

Exceptions are thrown when invalid lengths are presented to the server during BulkLoadBCP and RPC requests.

```
USHORTLEN_TYPE    =    BIGVARBINTYPE
                       /
                       BIGVARCHRTYPE
                       /
                       BIGBINARYTYPE
                       /
                       BIGCHARTYPE
                       /
                       NVARCHARTYPE
                       /
                       NCHARTYPE  ; the length value associated with
                                    these data types is specified
                                    within a USHORT

LONGLEN_TYPE      =    IMAGETYPE
/
NTEXTTYPE
/
SSVARIANTTYPE
/
TEXTTYPE
                       /
                       XMLTYPE    ; the length value associated with
                                    these data types is specified
                                    within a LONG
```

Notes

- MaxLength for an SSVARIANTTYPE is 8009 (8000 for strings). For more details, see section 2.2.5.5.4.
- XMLTYPE is only a valid LONGLEN_TYPE for BulkLoadBCP.

MaxLength for an SSVARIANTTYPE is 8009 (string of 8000 bytes).

```
VARLENTYPE        =    BYTELEN_TYPE
                       /
                       USHORTLEN_TYPE
                        /
                       LONGLEN_TYPE
```

Nullable values are returned by using the INTNTYPE, BITNTYPE, FLTNTYPE, GUIDTYPE, MONEYNTYPE, and DATETIMNTYPE tokens which will use the length byte to specify the length of the value or GEN_NULL as appropriate.

Copyright © 2014 Microsoft Corporation.

There are two types of variable-length data types. These are real variable-length data types, like char and binary, and nullable data types, which have either a normal fixed length that corresponds to their type or to a special length if null.

Char and binary data types have values that either are null or are 0 to 65534 (0x0000 to 0xFFFE) bytes of data. Null is represented by a length of 65535 (0xFFFF). A non-nullable char or binary can still have a length of zero (for example, an empty value). A program that MUST pad a value to a fixed length typically adds blanks to the end of a char and adds binary zeros to the end of a binary.

Text and image data types have values that either are null or are 0 to 2 gigabytes (0x00000000 to 0x7FFFFFFF bytes) of data. Null is represented by a length of -1 (0xFFFFFFFF). No other length specification is supported.

Other nullable data types have a length of 0 when they are null.

2.2.5.4.3 Partially Length-Prefixed Data Types

The data value corresponding to the set of data types defined in this section follows the rule defined in the partially length-prefixed stream definition (section 2.2.5.2.3).

```
PARTLENTYPE      =   XMLTYPE
                     /
                     BIGVARCHRTYPE
                     /
                     BIGVARBINTYPE
                     /
                     NVARCHARTYPE

                     /
                     UDTTYPE
```

BIGVARCHRTYPE, BIGVARBINTYPE, and NVARCHARTYPE can represent two types each:

- The regular type with a known maximum size range from 0 to 8000, defined by USHORTLEN_TYPE.
- A type with unlimited max size, known as varchar(max), varbinary(max) and nvarchar(max), which has a max size of 0xFFFF, defined by PARTLENTYPE. This class of types was introduced in TDS 7.2.

2.2.5.5 Data Type Details

The subsections within this section specify the formats in which values of system data types are serialized in TDS.

2.2.5.5.1 System Data Type Values

The subsections within this section specify the formats in which values of various common system data types are serialized in TDS.

2.2.5.5.1.1 Integers

All integer types are represented in reverse byte order (little-endian) unless otherwise specified. Each integer takes a whole number of bytes as follows:

Copyright © 2014 Microsoft Corporation.

bit: 1 byte

tinyint: 1 byte

smallint: 2 bytes

int: 4 bytes

bigint: 8 bytes

2.2.5.5.1.2 Timestamp

timestamp/rowversion is represented as an 8-byte binary sequence with no particular interpretation.

2.2.5.5.1.3 Character and Binary Strings

See Variable-Length Data Types (section 2.2.5.4.2) and Partially Length-Prefixed Data Types (section 2.2.5.4.3).

2.2.5.5.1.4 Fixed-Point Numbers

smallmoney is represented as a 4-byte signed integer. The TDS value is the **smallmoney** value multiplied by 10^4.

money is represented as an 8-byte signed integer. The TDS value is the **money** value multiplied by 10^4. The 8-byte signed integer itself is represented in the following sequence:

- One 4-byte integer that represents the more significant half.
- One 4-byte integer that represents the less significant half.

2.2.5.5.1.5 Floating-Point Numbers

float(*n*) follows the 32-bit [IEEE754] binary specification when *n* <= 24 and the 64-bit [IEEE754] binary specification when 25 <= *n* <= 53.

2.2.5.5.1.6 Decimal/Numeric

Decimal or Numeric is defined as **decimal**(*p*, *s*) or **numeric**(*p*, *s*), where *p* is the precision and *s* is the scale. The value is represented in the following sequence:

- One 1-byte unsigned integer that represents the sign of the decimal value as follows:
  - 0 means negative.
  - 1 means nonnegative.
- One 4-, 8-, 12-, or 16-byte signed integer that represents the decimal value multiplied by 10^s. The maximum size of this integer is determined based on *p* as follows:
  - 4 bytes if 1 <= *p* <= 9.
  - 8 bytes if 10 <= *p* <= 19.
  - 12 bytes if 20 <= *p* <= 28.

Copyright © 2014 Microsoft Corporation.

- 16 bytes if 29 <= *p* <= 38.

The actual size of this integer could be less than the maximum size, depending on the value. In all cases, the integer part MUST be either 4, 8, 12, or 16 bytes.

2.2.5.5.1.7 GUID

uniqueidentifier is represented as a 16-byte binary sequence with no specific interpretation.

2.2.5.5.1.8 Date/Times

smalldatetime is represented in the following sequence:

- One 2-byte unsigned integer that represents the number of days since January 1, 1900.
- One 2-byte unsigned integer that represents the number of minutes elapsed since 12 AM that day.

datetime is represented in the following sequence:

- One 4-byte signed integer that represents the number of days since January 1, 1900. Negative numbers are allowed to represent dates since January 1, 1753.
- One 4-byte unsigned integer that represents the number of one three-hundredths of a second (300 counts per second) elapsed since 12 AM that day.

date is represented as one 3-byte unsigned integer that represents the number of days since January 1, year 1.

time(*n*) is represented as one unsigned integer that represents the number of 10^{-n} second increments since 12 AM within a day. The length, in bytes, of that integer depends on the scale *n* as follows:

- 3 bytes if 0 <= *n* < = 2.
- 4 bytes if 3 <= *n* < = 4.
- 5 bytes if 5 <= *n* < = 7.

datetime2(*n*) is represented as a concatenation of **time**(*n*) followed by **date** as specified above.

datetimeoffset(*n*) is represented as a concatenation of **datetime2**(*n*) followed by one 2-byte signed integer that represents the time zone offset as the number of minutes from UTC. The time zone offset MUST be between -840 and 840.

2.2.5.5.2 Common Language Runtime (CLR) Instances

The following data type definition stream is used for UDT_INFO in TYPE_INFO. This data type was introduced in TDS 7.2.

```
DB_NAME                 =   B_VARCHAR  ; database name of the UDT
SCHEMA_NAME             =   B_VARCHAR  ; schema name of the UDT
TYPE_NAME               =   B_VARCHAR  ; type name of the UDT

MAX_BYTE_SIZE           =   USHORT     ; max length in bytes
ASSEMBLY_QUALIFIED_NAME =   US_VARCHAR ; name of the CLR assembly
```


Copyright © 2014 Microsoft Corporation.

```
UDT_METADATA            =   ASSEMBLY_QUALIFIED_NAME

UDT_INFO_IN_COLMETADATA =   MAX_BYTE_SIZE
                            DB_NAME
                            SCHEMA_NAME
                            TYPE_NAME
                            UDT_METADATA

UDT_INFO_IN_RPC  =    DB_NAME       ; database name of the UDT
                      SCHEMA_NAME   ; schema name of the UDT
                      TYPE_NAME     ; type name of the UDT

UDT_INFO            = UDT_INFO_IN_COLMETADATA        ;when sent as part of COLMETADATA
                        /
                        UDT_INFO_IN_RPC              ;when sent as part of RPC call
```

MAX_BYTE_SIZE is only sent from the server to the client in COLMETADATA and is an unsigned short with a value within the range 1 to 8000 or 0xFFFF. The value 0xFFFF signifies the maximum LOB size indicating a UDT with a maximum size greater than 8000 bytes (also referred to as a Large UDT; introduced in TDS 7.3). MAX_BYTE_SIZE is not sent to the server as part of RPC calls.

Note UserType in the COLMETADATA stream is always 0x0000 for UDTs. The actual data value format associated with a UDT data type definition stream is specified in [MS-SSCLRT].

2.2.5.5.3 XML Values

This section defines the XML data type definition stream, which was introduced in TDS 7.2.

```
SCHEMA_PRESENT= BYTE;
DBNAME                =   B_VARCHAR
OWNING_SCHEMA         =   B_VARCHAR
XML_SCHEMA_COLLECTION =   US_VARCHAR

XML_INFO              =   SCHEMA_PRESENT
                          [DBNAME OWNING_SCHEMA
                          XML_SCHEMA_COLLECTION]
```

SCHEMA_PRESENT specifies "0x01" if the type has an associated schema collection and DBNAME, OWNING_SCHEMA and XML_SCHEMA_COLLECTION MUST be included in the stream, or '0x00' otherwise.

DBNAME specifies the name of the database where the schema collection is defined.

OWNING_SCHEMA specifies the name of the relational schema containing the schema collection.

XML_SCHEMA_COLLECTION specifies the name of the XML schema collection to which the type is bound.

Note The actual data value format associated with a XML data type definition stream uses the [MS-BINXML] format. For more details, see [MS-BINXML].

2.2.5.5.4 SQL_VARIANT Values

The SSVARIANTTYPE is a special data type that acts as a place holder for other data types. When a SSVARIANTTYPE is filled with a data value, it takes on properties of the base data type that

Copyright © 2014 Microsoft Corporation.

represents the data value. To support this dynamic change, for those that are not NULL (GEN_NULL) the SSVARIANTTYPE instance has an SSVARIANT_INSTANCE internal structure according to the following definition.

```
VARIANT_BASETYPE    =   BYTE      ; data type definition
VARIANT_PROPBYTES   =   BYTE      ; see below
VARIANT_PROPERTIES  =   *BYTE     ; see below
VARIANT_DATAVAL     =   1*BYTE    ; actual data value

SSVARIANT_INSTANCE  =   VARIANT_BASETYPE
                        VARIANT_PROPBYTES
                        VARIANT_PROPERTIES
                        VARIANT_DATAVAL
```

VARIANT_BASETYPE is the TDS token of the base type.

VARIANT_BASETYPE	VARIANT_PROPBYTES	VARIANT_PROPERTIES
GUIDTYPE, BITTYPE, INT1TYPE, INT2TYPE, INT4TYPE, INT8TYPE, DATETIMETYPE, DATETIM4TYPE, FLT4TYPE, FLT8TYPE, MONEYTYPE, MONEY4TYPE, DATENTYPE	0	<not specified>
TIMENTYPE, DATETIME2NTYPE, DATETIMEOFFSETNTYPE	1	1 byte specifying scale
BIGVARBINTYPE, BIGBINARYTYPE	2	2 bytes specifying max length
NUMERICNTYPE, DECIMALNTYPE	2	1 byte for precision followed by 1 byte for scale
BIGVARCHRTYPE, BIGCHARTYPE, NVARCHARTYPE, NCHARTYPE	7	5-byte COLLATION, followed by a 2-byte max length

Note that data types cannot be NULL when inside a sql_variant. If the value is NULL, then the sql_variant itself SHOULD be NULL, but it is not allowed to specify a non-null sql_variant instance and have a NULL value wrapped inside it. A raw collation SHOULD NOT be specified within a sql_variant.<7>

2.2.5.5.5 Table Valued Parameter (TVP) Values

Table Valued Parameters (or User Defined Table Type as the type is known on the server) encapsulate an entire table of data with 1 to 1024 columns and an arbitrary number of rows. At the present time, TVPs are only permitted to be used as input parameters and do not appear in output parameters or in result set columns.

TVPs MUST only be sent by a TDS client reporting itself as a TDS major version 7.3 or later. If a client reporting itself less than TDS 7.3 attempts to send a TVP, this MUST result in the server rejecting the request with a TDS protocol error.

2.2.5.5.5.1 Metadata

```
TVPTYPE         =   %xF3
TVP_TYPE_INFO   =   TVPTYPE
                    TVP_TYPENAME
```


Copyright © 2014 Microsoft Corporation.

```
TVP_COLMETADATA
[TVP_ORDER_UNIQUE]
[TVP_COLUMN_ORDERING]
TVP_END_TOKEN
*TVP_ROW
TVP_END_TOKEN
```

Parameter	Description
TVPTYPE	%xF3
TVP_TYPENAME	Type name of the TVP
TVP_COLMETADATA	Column-specific metadata
[TVP_ORDER_UNIQUE]	Optional metadata token
[TVP_COLUMN_ORDERING]	Optional metadata token
TVP_END_TOKEN	End optional metadata
*TVP_ROW	0..N TVP_ROW tokens
TVP_END_TOKEN	End of rows

TVP_TYPENAME definition

```
DBNAME        =   B_VARCHAR ; Database where TVP type resides
OwningSchema  =   B_VARCHAR ; Schema where TVP type resides
TypeName      =   B_VARCHAR ; TVP type name
TVP_TYPENAME  =   DbName
                  OwningSchema
                  TypeName
```

TVP_COLMETADATA definition

```
DbName              =   B_VARCHAR ; Database where TVP type resides
fNullable           =   BIT       ; Column is nullable - %x01
fCaseSen            =   BIT       ; Column is case-sensitive - %x02
usUpdateable        =   2BIT      ; 2-bit value, one of:
                                  ; 0 = ReadOnly  - %x00
                                  ; 1 = ReadWrite - %x04
                                  ; 2 = Unknown   - %x08
fIdentity           =   BIT       ; Column is identity column - %x10
fComputed           =   BIT       ; Column is computed - %x20
usReservedODBC      =   2BIT      ; Reserved bits for ODBC - %x40+80
fFixedLenCLRType    =   BIT       ; Fixed length CLR type - %x100
fDefault            =   BIT       ; Column is default value - %x200
usReserved          =   6BIT      ; Six leftover reserved bits.

Flags               =   fNullable
                        fCaseSen
```


Copyright © 2014 Microsoft Corporation.

```
                        usUpdateable
                        fIdentity
                        fComputed
                        usReservedODBC
                        fFixedLenCLRType
                        fDefault
                        usReserved
Count               =   USHORT    ; Column count up to 1024 max
ColName             =   B_VARCHAR ; Name of column
UserType            =   ULONG     ; UserType of column

TypeColumnMetaData  =   UserType
                        Flags
                        TYPE_INFO
                        ColName   ; Column metadata instance



TVP_NULL_TOKEN      =   %xFFFF

TVP_COLMETADATA     =   TVP_NULL_TOKEN / (Count <Count>TvpColumnMetaData)
```

DbName, OwningSchema, and TypeName are limited to 128 Unicode characters max identifier length.

DbName is required to be zero-length, only OwningSchema and TypeName can be specified. DbName, OwningSchema, and TypeName are all optional fields and might ALL contain zero length strings. Client SHOULD follow these two rules:

- If the TVP is a parameter to a stored procedure or function where parameter metadata is available on the server side, then the client can send all zero-length strings for TVP_TYPENAME.
- If the TVP is a parameter to an ad-hoc SQL statement, parameter metadata information is not available on a stored procedure or function on the server. In this case the client is responsible to send sufficient type information with the TVP to allow the server to resolve the TVP type from sys.types. Failure to send needed type information in this case will result in complete failure of RPC call prior to execution.

Only one new flag, fDefault, is added here from existing COLMETADATA. ColName MUST be a zero-length string in the TVP.

Additional details about input TVPs and usage of flags

- For an input TVP, if the fDefault flag is set on a column, then the client MUST NOT emit the corresponding TvpColumnData data for the associated column when sending each TVP_ROW.
- For an input TVP, the fCaseSen, usUpdateable, and fFixedLenCLRType flags are ignored.
- usUpdateable is ignored by server on input, it is "calculated" metadata.
- The fFixedLenCLRType flag is not used by the server.
- Output TVPs are not currently supported.

TVP Flags Usage Chart

Copyright © 2014 Microsoft Corporation.

Flag	Input behavior
fNullable	Allowed
fCaseSen	Ignored
usUpdateable	Ignored
fIdentity	Allowed
fComputed	Allowed
usReservedODBC	Ignored
fFixedLenCLRType	Ignored
fDefault	Allowed (if set, data not sent in TvpColumnData)
usReserved	Ignored

2.2.5.5.5.2 Optional Metadata Tokens

TVP_ORDER_UNIQUE definition

```
TVP_ORDER_UNIQUE_TOKEN =   %x10
Count                  =   USHORT ; Count of ColNums to follow
ColNum                 =   USHORT ; A single-column ordinal
fOrderAsc              =   BIT    ; Column-ordered ascending - %x01
fOrderDesc             =   BIT    ; Column-ordered descending - %x02
fUnique                =   BIT    ; Column is in unique set - %x04
Reserved1              =   5BIT   ; Five reserved bits

OrderUniqueFlags       =   fOrderAsc
                           fOrderDesc
                           fUnique
                           Reserved1

TVP_ORDER_UNIQUE       =   TVP_ORDER_UNIQUE_TOKEN
                           (Count <Count>(ColNum OrderUniqueFlags))
```

TVP_ORDER_UNIQUE is similar to the ORDER token that is currently used in TDS responses from the server.

TVP_ORDER_UNIQUE is optional.

ColNum ordinals are 1..N, where 1 is the first column in TVP_COLMETADATA. That is, ordinals start with 1.

Each TVP_ORDER_UNIQUE token can describe a set of columns for ordering and/or a set of columns for uniqueness.

The first column ordinal with an ordering bit set is the primary sort column, the second column ordinal with an ordering bit set is the secondary sort column, and so on.

The client can send 0 or 1 TVP_ORDER_UNIQUE tokens in a single TVP.

Copyright © 2014 Microsoft Corporation.

The TVP_ORDER_UNIQUE token MUST always be sent after TVP_COLMETADATA and before the first TVP_ROW token.

When a TVP is sent to the server, each ColNum ordinal inside a TVP_ORDER_UNIQUE token MUST refer to a client generated column. Ordinals that refer to columns with fDefault set will be rejected by the server.

OrderUniqueFlags Possible Combinations And Meaning

fOrderAsc	fOrderDesc	fUnique	Meaning
FALSE	FALSE	FALSE	Invalid flag state, rejected by server
FALSE	FALSE	TRUE	Column is in unique set
FALSE	TRUE	FALSE	Column is ordered descending
FALSE	TRUE	TRUE	Column is ordered descending and in unique set
TRUE	FALSE	FALSE	Column is ordered ascending
TRUE	FALSE	TRUE	Column is ordered ascending and in unique set
TRUE	TRUE	FALSE	Invalid flag state, rejected by server
TRUE	TRUE	TRUE	Invalid flag state, rejected by server

TVP_COLUMN_ORDERING

TVP_COLUMN_ORDERING is an optional TVP metadata token that is used to allow the TDS client to send a different ordering of the columns in a TVP from the default ordering.

ColNum ordinals are 1..N where 1 is first column in the TVP (ordinals start with 1 in other words). These are the same ordinals used with the TDS ORDER token, for example, to refer to column ordinal as the columns appear in left to right order.

```
TVP_COLUMN_ORDERING_TOKEN =   %x11
Count                     =   USHORT ; Count of ColNums to follow
ColNum                    =   USHORT ; A single-column ordinal

TVP_COLUMN_ORDERING       =   TVP_COLUMN_ORDERING_TOKEN
                              (Count <Count>ColNum)
```

The client can send 0 or 1 TVP_COLUMN_ORDERING tokens in a single TVP.

The TVP_COLUMN_ORDERING token MUST always be sent after TVP_COLMETADATA and before the first TVP_ROW token.

Additional details about TVP_COLUMN_ORDERING

TVP_COLUMN_ORDERING is used to re-order the columns in a TVP. For example if a TVP is defined as:

```
create type myTvpe as table (f1 int, f2 varchar(max), f3 datetime)
```


Copyright © 2014 Microsoft Corporation.

The TDS client might want to send the f2 field last inside the TVP as an optimization (streaming the large value last). So the client can send TVP_COLUMN_ORDERING with order 1,3,2 to indicate that inside the TVP_ROW section the column f1 is sent first, f3 is sent second, and f2 is sent third.

So the TVP_COLUMN_ORDERING token on the wire for this example would be:

```
11    ; TVP_COLUMN_ORDERING_TOKEN
03 00 ; Count  - Number of ColNums to follow.
01 00 ; ColNum - TVP column ordinal 1 is sent first in TVP_COLMETADATA.
03 00 ; ColNum - TVP column ordinal 3 is sent second in TVP_COLMETADATA.
02 00 ; ColNum - TVP column ordinal 2 is sent third in TVP_COLMETADATA.
```

Duplicate ColNum values are considered an error condition. The ordinal values of the columns in the actual TVP type are ordered starting with 1 for the first column and adding one for each column from left to right. The client MUST send one ColNum for each column described in the TVP_COLMETADATA (so Count MUST match number of columns in TVP_COLMETADATA).

TVP_ROW definition

```
TVP_ROW_TOKEN  =   %x01           ;  A row as defined by TVP_COLMETADATA follows
TvpColumnData  =   TYPE_VARBYTE   ;  Actual value must match metadata for the column
AllColumnData  =   *TvpColumnData ;  Chunks of data, one per non-default column defined in
TVP_COLMETADATA.
TVP_ROW        =   TVP_ROW_TOKEN
                   AllColumnData
TVP_END_TOKEN  =   %x00           ;  Terminator tag for TVP type meaning no moreTVP_ROWs to
follow and end of successful transmission of a single TVP.
```

TvpColumnData is repeated once for each non-default column of data defined in TVP_COLMETADATA.

Each row will contain one data "cell" per column specified in TVP_COLMETADATA. On input, columns with the fDefault flag set in TVP_COLMETADATA will be skipped to avoid sending redundant data.

Column data is ordered in same order as the order of items defined in TVP_COLMETADATA unless a TVP_COLUMN_ORDERING token has been sent to indicate a change in the ordering of the row values.

2.2.5.5.5.3 TDS Type Restrictions

Within a TVP, the following legacy TDS types are not supported:

TDS type	Replacement type
Binary	BigBinary
VarBinary	BigVarBinary

Copyright © 2014 Microsoft Corporation.

TDS type	Replacement type
Char	BigChar
VarChar	BigVarChar
Bit	BitN
Int1	IntN
Int2	IntN
Int4	IntN
Int8	IntN
Float4	FloatN
Float8	FloatN
Money	MoneyN
Decimal	DecimalN
Numeric	NumericN
DateTime	DatetimeN
DateTime4	DatetimeN
Money4	MoneyN

Additional types not allowed in TVP:

- Null type (NULLTYPE:='0x1f') is not allowed in a TVP.
- TVP type is not allowed in a TVP (no nesting of TVP in a TVP).
- TDS types should not be confused with data types for a database server that supports SQL.

2.2.5.6 Type Info Rule Definition

The TYPE_INFO rule applies to several messages used to describe column information. For columns of fixed data length, the type is all that is required to determine the data length. For columns of a variable-length type, TYPE_VARLEN defines the length of the data contained within the column, with the following exceptions introduced in TDS 7.3:

DATE MUST NOT have a TYPE_VARLEN. The value is either 3 bytes or 0 bytes (null).

TIME, DATETIME2, and DATETIMEOFFSET MUST NOT have a TYPE_VARLEN. The lengths are determined by the SCALE as indicated in section 2.2.5.4.2.

PRECISION and SCALE MUST occur if the type is NUMERIC, NUMERICN, DECIMAL, or DECIMALN.

SCALE (without PRECISION) MUST occur if the type is TIME, DATETIME2, or DATETIMEOFFSET (introduced in TDS 7.3). PRECISION MUST be less than or equal to decimal 38 and SCALE MUST be less than or equal to the precision value.

Copyright © 2014 Microsoft Corporation.

COLLATION occurs only if the type is BIGCHARTYPE, BIGVARCHRTYPE, TEXTTYPE, NTEXTTYPE, NCHARTYPE, or NVARCHARTYPE.

UDT_INFO always occurs if the type is UDTTYPE.

XML_INFO always occurs if the type is XMLTYPE.

USHORTMAXLEN does not occur if PARTLENTYPE is XMLTYPE or UDTTYPE.

```
USHORTMAXLEN     =   %xFFFF

TYPE_INFO        =   FIXEDLENTYPE
                     /
                     (VARLENTYPE TYPE_VARLEN [COLLATION])
                     /
                     (VARLENTYPE TYPE_VARLEN [PRECISION SCALE])
                     /
                     (VARLENTYPE SCALE) ; (introduced in TDS 7.3)
                     /
                     VARLENTYPE         ; (introduced in TDS 7.3)
                     /
                     (PARTLENTYPE
                     [USHORTMAXLEN]
                     [COLLATION]
                     [XML_INFO]
                     [UDT_INFO])
```

2.2.5.7 Data Buffer Stream Tokens

The tokens defined as follows are used as part of the token-based data stream. Details about how each token is used inside the data stream are in section 2.2.6.

```
ALTMETADATA_TOKEN    =   %x88
ALTROW_TOKEN         =   %xD3
COLMETADATA_TOKEN    =   %x81
COLINFO_TOKEN        =   %xA5
DONE_TOKEN           =   %xFD
DONEPROC_TOKEN       =   %xFE
DONEINPROC_TOKEN     =   %xFF
ENVCHANGE_TOKEN      =   %xE3
ERROR_TOKEN          =   %xAA
FEATUREEXTACK_TOKEN  =   %xAE ; (introduced in TDS 7.4)
INFO_TOKEN           =   %xAB
LOGINACK_TOKEN       =   %xAD
NBCROW_TOKEN         =   %xD2 ; (introduced in TDS 7.3)
OFFSET_TOKEN         =   %x78
ORDER_TOKEN          =   %xA9
RETURNSTATUS_TOKEN   =   %x79
RETURNVALUE_TOKEN    =   %xAC
ROW_TOKEN            =   %xD1
SESSIONSTATE_TOKEN   =   %xE4 ; (introduced in TDS 7.4)
SSPI_TOKEN           =   %xED
TABNAME_TOKEN        =   %xA4
```


Copyright © 2014 Microsoft Corporation.

2.2.6 Packet Header Message Type Stream Definition

2.2.6.1 Bulk Load BCP

Stream Name:

```
BulkLoadBCP
```

Stream Function:

Describes the format of bulk-loaded data through the **"INSERT BULK"** T-SQL statement. The format is a COLMETADATA token describing the data being sent, followed by multiple ROW tokens, ending with a DONE token. The stream is equivalent to that produced by the server if it were sending the same rowset on output.

Stream Comments:

- Packet header type is 0x07.
- This message sent to the server contains bulk data to be inserted. The client MUST have previously notified the server where this data is to be inserted. For more information about the INSERT BULK syntax, see [MSDN-INSERT].
- A sample BulkLoadBCP message is in section 4.12.

Stream-Specific Rules:

```
BulkLoad_METADATA    =    COLMETADATA_TOKEN
BulkLoad_ROW         =    ROW_TOKEN
BulkLoad_DONE        =    DONE_TOKEN
```

Submessage Definition:

```
BulkLoadBCP          =    BulkLoad_METADATA
                          *BulkLoad_ROW
                          BulkLoad_DONE
```

Note that for INSERT BULK operations, XMLTYPE is to be sent as NVARCHAR(N) or NVARCHAR(MAX) data type. An error is produced if XMLTYPE is specified.

INSERT BULK operations for data type UDTTYPE is not supported. Use VARBINARYTYPE to **insert** instances of User Defined Types.

INSERT BULK operations do not support type specifications of DECIMALTYPE and NUMERICTYPE. To insert these data types, use DECIMALN and NUMERICNTYPE.

2.2.6.2 Bulk Load Update Text/Write Text

Stream Name:

Copyright © 2014 Microsoft Corporation.

```
BulkLoadUTWT
```

Stream Function:

Describes the format of bulk-loaded data with UpdateText or WriteText. The format is the length of the data followed by the data itself.

Stream Comments:

- Packet header type 0x07.
- This message sent to the server contains bulk data to be inserted. The client MUST have previously issued a **"WRITETEXT BULK"** or **"UPDATETEXT BULK"** T-SQL statement to the server. For information about the WRITETEXT BULK and UPDATETEXT BULK syntax, see [MSDN-WRITETEXT] and [MSDN-UPDATETEXT], respectively.
- The server returns a RETURNVALUE token containing the new timestamp for this column.

Stream-Specific Rules:

```
BulkData          =    L_VARBYTE
```

Sub Message Definition:

```
BulkLoadUTWT      =    BulkData
```

Stream Parameter Details

Parameter	Description
BulkData	Contains the BulkData length and BulkData data within the L_VARBYTE.

2.2.6.3 LOGIN7

Stream Name:

```
LOGIN7
```

Stream Function:

Defines the authentication rules for use between client and server.

Stream Comments:

- Packet header type 0x10.
- The length of a LOGIN7 stream MUST NOT be longer than 128K-1(byte) bytes.
- The OffsetLength and Data rules define the variable-length portions of this data stream. The OffsetLength rule lists the offset from the start of the structure, and the length for each

Copyright © 2014 Microsoft Corporation.

parameter. If the parameter is not used, the parameter length field MUST be 0. The data itself (for example, the Data rule) follows these parameters.

- The first parameter of the OffsetLength rule (ibHostName) indicates the start of the variable length portion of this data stream. As such it MUST NOT be 0. This is required for forward compatibility (for example, later versions of TDS, with additional parameters, can be successfully skipped by down-level servers).
- A sample LOGIN7 message is in section 4.2.

Stream-Specific Rules:

```
Length           =   DWORD
TDSVersion       =   DWORD
PacketSize       =   DWORD
ClientProgVer    =   DWORD
ClientPID        =   DWORD
ConnectionID     =   DWORD

fByteorder       =   BIT
fChar            =   BIT
fFloat           =   2BIT
fDumpLoad        =   BIT
fUseDB           =   BIT
fDatabase        =   BIT
fSetLang         =   BIT

OptionFlags1     =   fByteorder
                     fChar
                     fFloat
                     fDumpLoad
                     fUseDB
                     fDatabase
                     fSetLang

fLanguage        =   BIT
fODBC            =   BIT
fTranBoundary    =   BIT           ; (removed in TDS 7.2)
fCacheConnect    =   BIT           ; (removed in TDS 7.2)
fUserType        =   3BIT
fIntSecurity     =   BIT

OptionFlags2     =   fLanguage
                     fODBC
                     (fTransBoundary / FRESERVEDBIT)
                     (fCacheConnect / FRESERVEDBIT)
                     fUserType
                     fIntSecurity

fSQLType         =   4BIT
fOLEDB           =   BIT            ; (introduced in TDS 7.2)
fReadOnlyIntent  =   BIT            ; (introduced in TDS 7.4)

TypeFlags        =   fSQLType
                     (FRESERVEDBIT / fOLEDB)
                     (FRESERVEDBIT / fReadOnlyIntent)
                     2FRESERVEDBIT

fChangePassword  =   BIT            ; (introduced in TDS 7.2)
```


Copyright © 2014 Microsoft Corporation.

```
fUserInstance    =   BIT           ; (introduced in TDS 7.2)
fSendYukonBinaryXML =  BIT         ; (introduced in TDS 7.2)
fUnknownCollationHandling =  BIT   ; (introduced in TDS 7.3)
fExtension       =   BIT           ; (introduced in TDS 7.4)

OptionFlags3     =   (FRESERVEDBIT / fChangePassword)
                     (FRESERVEDBIT / fSendYukonBinaryXML)
                     (FRESERVEDBIT / fUserInstance)
                     (FRESERVEDBIT / fUnknownCollationHandling)
                     (FRESERVEDBIT / fExtension)
                     3FRESERVEDBIT

ClientTimZone    =   LONG;
ClientLCID       =   LCID
                     ColFlags
                     Version

ibHostName       =   USHORT
cchHostName      =   USHORT
ibUserName       =   USHORT
cchUserName      =   USHORT
ibPassword       =   USHORT
cchPassword      =   USHORT
ibAppName        =   USHORT
cchAppName       =   USHORT
ibServerName     =   USHORT
cchServerName    =   USHORT
ibUnused         =   USHORT
cbUnused         =   USHORT
ibExtension      =   USHORT    ; (introduced in TDS 7.4)
cbExtension      =   USHORT    ; (introduced in TDS 7.4)
ibCltIntName     =   USHORT
cchCltIntName    =   USHORT
ibLanguage       =   USHORT
cchLanguage      =   USHORT
ibDatabase       =   USHORT
cchDatabase      =   USHORT
ClientID         =   6BYTE
ibSSPI           =   USHORT
cbSSPI           =   USHORT
ibAtchDBFile     =   USHORT
cchAtchDBFile    =   USHORT
ibChangePassword =   USHORT    ; (introduced in TDS 7.2)
cchChangePassword =  USHORT    ; (introduced in TDS 7.2)
cbSSPILong       =   DWORD     ; (introduced in TDS 7.2)

OffsetLength     =   ibHostName
                     cchHostName
                     ibUserName
                     cchUserName
                     ibPassword
                     cchPassword
                     ibAppName
                     cchAppName
                     ibServerName
                     cchServerName
                     (ibUnused / ibExtension)
                     (cchUnused / cbExtension)
                     ibCltIntName
```


Copyright © 2014 Microsoft Corporation.

```
                    cchCltIntName
                    ibLanguage
                    cchLanguage
                    ibDatabase
                    cchDatabase
                    ClientID
                    ibSSPI
                    cbSSPI
                    ibAtchDBFile
                    cchAtchDBFile
                    ibChangePassword
                    cchChangePassword
                    cbSSPILong
```

All variable-length fields in the login record are optional. This means that the length of the field can be specified as 0. If the length is specified as 0, then the offset MUST be ignored. The only exception is ibHostName, which MUST always point to the beginning of the variable-length data in the login record even in the case where no variable-length data is included.

```
Data             =   *BYTE

FeatureId        =   BYTE
FeatureDataLen   =   DWORD
FeatureData      =   *BYTE

TERMINATOR       =   %xFF       ; signal of end of feature option

FeatureOpt       =   (FeatureId
                     FeatureDataLen
                     FeatureData)
                     /
                     TERMINATOR

FeatureExt       =   1*FeatureOpt        ; (introduced in TDS 7.4)
```

Stream Definition:

```
LOGIN7           =   Length
                     TDSVersion
                     PacketSize
                     ClientProgVer
                     ClientPID
                     ConnectionID
                     OptionFlags1
                     OptionFlags2
                     TypeFlags
                     (FRESERVEDBYTE / OptionFlags3)
                     ClientTimZone
                     ClientLCID
                     OffsetLength
                     Data
                     [FeatureExt]
```

Stream Parameter Details

Copyright © 2014 Microsoft Corporation.

Parameter	Description
Length	The total length of the LOGIN7 structure.
TDSVersion	The highest TDS version being used by the client (for example, 0x00000071 for TDS 7.1). If the TDSVersion value sent by the client is greater than the value that the server recognizes, the server MUST use the highest TDS version that it can use. This provides a mechanism for clients to discover the server TDS by sending a standard LOGIN7 message. If the TDSVersion value sent by the client is lower than the highest TDS version the server recognizes, the server MUST use the TDS version sent by the client.<8> For information about what the server sends to the client, see the LOGINACK token.
PacketSize	The desired packet size being requested by the client.
ClientProgVer	The version of the interface library (for example, ODBC or OLEDB) being used by the client.
ClientPID	The process ID of the client application.
ConnectionID	The connection ID of the primary Server. Used when connecting to an "Always Up" backup server.

OptionFlags1	▪ Represented in least significant bit order.
	▪ fByteOrder: The byte order used by client for numeric and datetime data types.
	  ▪ 0 = ORDER_X86
	  ▪ 1 = ORDER_68000
	▪ fChar: The character set used on the client.
	  ▪ 0 = CHARSET_ASCII
	  ▪ 1 = CHARSET_EBDDIC
	▪ fFloat: The type of floating point representation used by the client.
	  ▪ 0 = FLOAT_IEEE_754
	  ▪ 1 = FLOAT_VAX
	  ▪ 2 = ND5000
	▪ fDumpLoad: Set is dump/load or BCP capabilities are needed by the client.
	  ▪ 0 = DUMPLOAD_ON
	  ▪ 1 = DUMPLOAD_OFF
	▪ fUseDB: Set if the client desires warning messages on execution of the USE SQL statement. If this flag is not set, the server MUST NOT inform the client when the database changes, and therefore the client will be unaware of any accompanying collation changes.
	  ▪ 0 = USE_DB_OFF
	  ▪ 1 = USE_DB_ON

Copyright © 2014 Microsoft Corporation.

Parameter	Description

	▪ fDatabase: Set if the change to initial database needs to succeed if the connection is to succeed.
	  ▪ 0 = INIT_DB_WARN
	  ▪ 1 = INIT_DB_FATAL
	▪ fSetLang: Set if the client desires warning messages on execution of a language change statement.
	  ▪ 0 = SET_LANG_OFF
	  ▪ 1 = SET_LANG_ON

OptionFlags2	▪ Represented in least significant bit order.
	▪ fLanguage: Set if the change to initial language needs to succeed if the connect is to succeed.
	  ▪ 0 = INIT_LANG_WARN
	  ▪ 1 = INIT_LANG_FATAL
	▪ fODBC: Set if the client is the ODBC driver. This causes the server to set ANSI_DEFAULTS to ON, CURSOR_CLOSE_ON_COMMIT and IMPLICIT_TRANSACTIONS to OFF, TEXTSIZE to 0x7FFFFFFF (2GB) (TDS 7.2 and earlier), TEXTSIZE to infinite (introduced in TDS 7.3), and ROWCOUNT to infinite.
	  ▪ 0 = ODBC_OFF
	  ▪ 1 = ODBC_ON
	▪ fTransBoundary
	▪ fCacheConnect
	▪ fUserType: The type of user connecting to the server.
	  ▪ 0 = USER_NORMAL—regular logins
	  ▪ 1 = USER_SERVER—reserved
	  ▪ 2 = USER_REMUSER—Distributed Query login
	  ▪ 3 = USER_SQLREPL—replication login
	▪ fIntSecurity: The type of security required by the client.
	  ▪ 0 = INTEGRATED_SECURTY_OFF
	  ▪ 1 = INTEGRATED_SECURITY_ON

TypeFlags	▪ Represented in least significant bit order. ▪ fSQLType: The type of SQL the client sends to the server. ▪ 0 = SQL_DFLT

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	▪ 1 = SQL_TSQL ▪ fOLEDB: Set if the client is the OLEDB driver. This causes the server to set ANSI_DEFAULTS to ON, CURSOR_CLOSE_ON_COMMIT and IMPLICIT_TRANSACTIONS to OFF, TEXTSIZE to 0x7FFFFFFF (2GB) (TDS 7.2 and earlier), TEXTSIZE to infinite (introduced in TDS 7.3), and ROWCOUNT to infinite. ▪ 0 = OLEDB_OFF ▪ 1 = OLEDB_ON ▪ fReadOnlyIntent: This bit is introduced in TDS 7.4; however, TDS 7.1, 7.2, and 7.3 clients can also use this bit in LOGIN7 to specify that the application intent of the connection is read-only. The server SHOULD ignore this bit if the highest TDS version supported by the server is lower than TDS 7.4.
OptionFlags3	▪ Represented in least significant bit order. ▪ fChangePassword: Specifies whether the login request SHOULD change password. ▪ 0 = No change request. ibChangePassword MUST be 0. ▪ 1 = Request to change login's password. ▪ fSendYukonBinaryXML: 1 if XML data type instances are returned as binary XML. ▪ fUserInstance: 1 if client is requesting separate process to be spawned as user instance. ▪ fUnknownCollationHandling: This bit is used by the server to determine if a client is able to properly handle collations introduced after TDS 7.2. TDS 7.2 and earlier clients are encouraged to use this login packet bit. Servers MUST ignore this bit when it is sent by TDS 7.3 or 7.4 clients. See [MSDN-SQLCollation] and [MS-LCID] documents for the complete list of collations for a database server that supports SQL and LCIDs. ▪ 0 = The server MUST restrict the collations sent to a specific set of collations. It MAY disconnect or send an error if some other value is outside the specific collation set. The client MUST properly support all collations within the collation set. ▪ 1 = The server MAY send any collation that fits in the storage space. The client MUST be able to both properly support collations and gracefully fail for those it does not support. ▪ fExtension: Specifies whether ibExtension/cbExtension fields are used. ▪ 0 = ibExtension/cbExtension fields are not used. The fields are treated the same as ibUnused/cchUnused. ▪ 1 = ibExtension/cbExtension fields are used.
ClientTimeZone	The time zone of the client machine.
ClientLCID	The language code identifier (LCID) value for the client collation. If ClientLCID is specified, the specified collation is set as the session collation. Note that the total ClientLCID is 4 bytes, which implies that there is no support for SQL Sort orders.

Copyright © 2014 Microsoft Corporation.

Parameter	Description

OffsetLength	The variable portion of this message. A stream of bytes in the order shown, indicates the offset (from the start of the message) and length of various parameters:
	▪ IbHostname & cchHostName: The client machine name.
	▪ IbUserName & cchUserName: The client user ID.
	▪ IbPassword & cchPassword: The password supplied by the client.
	▪ IbAppName & cchAppName: The client application name.
	▪ IbServerName & cchServerName: The server name.
	▪ ibUnused & cbUnused: These parameters were reserved until TDS 7.4.
	▪ ibExtension & cbExtension: This points to an extension block. Introduced in TDS 7.4 when fExtension is 1. The content pointed by ibExtension is defined as follows:
	`ibFeatureExtLong = DWORD`
	`Extension = ibFeatureExtLong`
	ibFeatureExtLong provides the offset (from the start of the message) of FeatureExt block. ibFeatureExtLong MUST be 0 if FeatureExt block does not exist.
	Extension block can be extended in future. The client MUST NOT send more data than needed. The server SHOULD ignore any appended data that is unknown to the server.
	▪ ibCltIntName & cchCltIntName: The interface library name (ODBC or OLEDB).
	▪ ibLanguage & cchLanguage: The initial language (overrides the user ID's default language).
	▪ ibDatabase & cchDatabase: The initial database (overrides the user ID's default database).
	▪ ClientID: The unique client ID (created used NIC address).
	▪ ibSSPI & cbSSPI: SSPI data.
	If cbSSPI < USHRT_MAX, then this length MUST be used for SSPI and cbSSPILong MUST be ignored.
	If cbSSPI == USHRT_MAX, then cbSSPILong MUST be checked.
	If cbSSPILong > 0, then that value MUST be used. If cbSSPILong ==0, then cbSSPI (USHRT_MAX) MUST be used.
	▪ ibAtchDBFile & cchAtchDBFile: The file name for a database that is to be attached during the connection process.
	▪ ibChangePassword & cchChangePassword: New password for the specified login. Introduced in TDS 7.2.
	▪ cbSSPILong: Used for large SSPI data when cbSSPI==USHRT_MAX. Introduced in

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	TDS 7.2.
Data	The actual variable-length data portion referred to by OffsetLength.
FeatureId	The unique identifier number of a feature. See detailed description in the table below.
FeatureDataLen	The length, in bytes, of FeatureData for the corresponding FeatureID.
FeatureData	Data of the feature. Each feature defines its own data format. Data for existing features are defined in the table below.
FeatureExt	The data block that can be used to inform and/or negotiate features between client and server. It contains data for one or more optional features. Each feature is assigned an identifier, followed by data length and data. The data for each feature is defined by the feature's own logic. If the server does not support the specific feature, it MUST skip the feature data and jump to next feature. If needed, each feature SHOULD have its own logic to detect if the server accepts the feature option. Optionally, a feature can use FeatureExtAck token to acknowledge the feature along with LOGINACK. The detailed acknowledge data SHOULD be defined by the feature itself.

FeatureExt Feature Option and Description

FeatureId	FeatureData Description

%0x01
(SESSIONRECOVERY)
	Session Recovery feature. This feature is used to recover the session state of a previous connection. Content is defined as follows:
	
	`Length = DWORD`
	`RecoveryDatabase = B_VARCHAR`
	`RecoveryCollation = BYTELEN [COLLATION]`
	`RecoveryLanguage = B_VARCHAR`
	
	`SessionRecoveryData = Length`
	`RecoveryDatabase`
	`RecoveryCollation`
	`RecoveryLanguage`
	`SessionStateDataSet`
	
	`InitSessionRecoveryData = SessionRecoveryData`
	`SessionRecoveryDataToBe = SessionRecoveryData`
	
	`FeatureData = [InitSessionRecoveryData`
	`SessionRecoveryDataToBe]`
	
	The Length field is the length, in bytes, of SessionRecoveryData excluding the Length field itself. SessionStateDataSet is described in section 2.2.7.19. The length of SessionStateDataSet can be derived from the Length field and the length of RecoveryDatabase, RecoveryCollation, and RecoveryLanguage. The maximum length for RecoveryDatabase and RecoveryLanguage is 128 UNICODE characters.
	There are two sets of SessionRecoveryData. The data for the first set, InitSessionRecoveryData, SHOULD come from the initial login response data of

Copyright © 2014 Microsoft Corporation.

FeatureId	FeatureData Description
	the initial connection to be recovered, specifically, the Database/Collation/Language ENVCHANGE data and SessionStateDataSet in FeatureExtAck. Data for the second set, SessionRecoveryDataToBe, SHOULD come from the latest ENVCHANGE for Database/Collation/Language from the connection to be recovered and the latest data for each StateId in SessionStateData from the connection to be recovered. If login succeeded on this recovery connection, the session state of the connection MUST be set to SessionRecoveryDataToBe. To save space, if data for RecoveryDatabase/RecoveryCollation/RecoveryLanguage in SessionRecoveryDataToBe is the same as data in InitSessionRecoveryData, the length value of each field SHOULD be 0. If data for any session StateId is unchanged from InitSessionRecoveryData, the corresponding StateId data SHOULD be skipped in SessionRecoveryDataToBe. When this feature option is received and the server supports connection recovery, a FeatureExtAck token that contains data for SESSIONRECOVERY feature MUST be returned along with LOGINACK in the login response to indicate that the server supports the feature. If SESSIONRECOVERY was not acknowledged in the login response, the server does not support the feature and the client MUST disable the feature for this connection. The client can request this feature option with zero FeatureDataLen. This is used during login for the initial connection to indicate that the client prefers this feature. When the client sends this feature option with non-zero FeatureDataLen during login, the option data SHOULD come from a previous connection. The TDS version in the login request MUST be the same as the TDS version negotiated for the connection to be recovered. The server MUST return the same TDS version in the login response, and if not, the client MUST disconnect the connection and raise an error to the upper layer. If a login record with non-zero FeatureDataLen of this feature is received and the server supports this feature, the server MUST: ▪ Force TDS version negotiation to use the TDS version requested by the client, and fail the login if the requested TDS version is not known to the server, for example, a TDS version that is later than the highest one currently on the server. ▪ Validate the content in SessionRecoveryData, and fail the login if any data is invalid or any unknown session state exists. Once the feature is negotiated to be enabled, the server SHOULD send session state updates to the client via a SESSIONSTATE token during the lifetime of the connection. The client MUST track the initial session state data as well as the latest session state data. Session state data is updated via a SESSIONSTATE token incrementally. When a client requests RESETCONNECTION/RESETCONNECTIONSKIPTRAN and the server acknowledges the request, both the client and the server MUST update the baseline of the session state data to be the same as the initial state as defined by InitSessionRecoveryData, and any further state update SHOULD be on top of the initial state. Session state data can be used to recover a dead connection as defined by SessionRecoveryData. The client SHOULD try to recover a dead connection if the latest fRecovery bit is TRUE for all StateId received from the server. The client MUST NOT try to recover a dead connection if the any latest fRecovery bit is FALSE.
%0x02 (FEDAUTH)	Presence of the FEDAUTH FeatureExt indicates that the client is authenticating by federated authentication. If this FeatureId is present, the value of fIntSecurity

Copyright © 2014 Microsoft Corporation.

FeatureId	FeatureData Description

	MUST be 0. The format of the data is as described below based on the bFedAuthLibrary that is used.
	When the bFedAuthLibrary is Live ID Compact Token, the format is as follows:
	
	`bFedAuthLibrary = 7BIT`
	`fFedAuthEcho = BIT`
	`Options = bFedAuthLibrary`
	`fFedAuthEcho`
	
	`FedAuthToken = L_VARBYTE`
	
	`Nonce = 32BYTE`
	`ChannelBindingToken = BYTESTREAM`
	`Signature = 32BYTE`
	
	`SignedData = Nonce`
	`[ChannelBindingToken]`
	`Signature`
	
	`FeatureData = Options`
	`FedAuthToken`
	`SignedData`
	
	bFedAuthLibrary: 7 bits, collectively treated as a 7-bit unsigned integer, indicating the library that is used by the client for federated authentication.
	0x00 = Live ID Compact Token. The format of the Live ID Compact Token and the way in which the Live ID Compact Token is obtained are out of the scope of this document.
	fFedAuthEcho: The intention of this flag is for the client to echo the server's FEDAUTHREQUIRED prelogin option, so that the server can validate that it was not tampered with. When the FederatedAuth field is present, the client MUST assign this flag to 1 if and only if the server's PRELOGIN response contained a FEDAUTHREQUIRED option with a B_FEDAUTHREQUIRED value of 0x01.
	FedAuthToken: The binary authentication token generated by the specified federated authentication library. The length of FedAuthToken MUST NOT be 0.
	Nonce: The nonce provided by the server during the Prelogin exchange, echoed back to the server by the client.
	ChannelBindingToken: This optional field MAY be omitted, but if encryption is being used for the lifetime of the TDS connection and the client is able to generate a channel binding token, the field SHOULD be included in the payload. When present, ChannelBindingToken contains the channel binding token associated with the underlying SSL stream.
	Signature: The HMAC-SHA-256 [RFC6234] hash of the server-specified nonce and, if it is present in the FeatureData, the ChannelBindingToken, is generated by using the session key retrieved from the federated authentication context as the shared secret.
	The length of the ChannelBindingToken field is not explicitly conveyed in the protocol, but can be determined by comparing the FeatureDataLen against the length of the remainder of the feature data, which is explicitly transmitted in the protocol.
	When the bFedAuthLibrary is Security Token, the format is as follows:
	
	`bFedAuthLibrary = 7BIT`
	`fFedAuthEcho = BIT`
	`Options = bFedAuthLibrary`

Copyright © 2014 Microsoft Corporation.

FeatureId	FeatureData Description
	fFedAuthEcho FedAuthToken = L_VARBYTE Nonce = 32BYTE OtherData = Nonce FeatureData = Options FedAuthToken [OtherData] bFedAuthLibrary: 7 bits, collectively treated as a 7-bit unsigned integer, indicating the library that is used by the client for federated authentication. 0x01 = Security Token. The format of the token and the way in which this token is obtained are out of the scope of this document. fFedAuthEcho: The intention of this flag is for the client to echo the server's FEDAUTHREQUIRED prelogin option, so that the server can validate that it was not tampered with. When the FederatedAuth field is present, the client MUST assign this flag to 1 if and only if the server's PRELOGIN response contained a FEDAUTHREQUIRED option with a B_FEDAUTHREQUIRED value of 0x01. FedAuthToken: The binary authentication token generated by the specified federated authentication library. The length of FedAuthToken MUST NOT be 0. Nonce: The nonce provided by the server during the Prelogin exchange, echoed back to the server by the client. This field MUST be present if the server's PRELOGIN message included NONCE field. Otherwise, this field MUST NOT be present. All other values of bFedAuthLibrary are reserved.
%xFF (TERMINATOR)	This is the last option in FeatureExt.

Login Data Validation Rules

cchHostName MUST specify at most 128 Unicode characters.

cchUserName MUST specify at most 128 Unicode characters.

cchPassword MUST specify at most 128 Unicode characters.

cchAppName MUST specify at most 128 Unicode characters.

cchServerName MUST specify at most 128 Unicode characters.

cbExtension MUST NOT exceed 255 bytes.

cchCltIntName MUST specify at most 128 Unicode characters.

cchLanguage MUST specify at most 128 Unicode characters.

cchDatabase MUST specify at most 128 Unicode characters.

cchAtchDBFile MUST specify at most 260 Unicode characters.

cchChangePassword MUST specify at most 128 Unicode characters.

Copyright © 2014 Microsoft Corporation.

The value at ibUserName—if specified—is semantically enclosed in brackets ([]) and MUST conform to the rules for valid delimited object identifiers. Login MUST fail otherwise.

The value at ibDatabase—if specified—is semantically enclosed in brackets ([]) and MUST conform to the rules for valid delimited object identifiers. Login MUST fail otherwise.

Before submitting a password from the client to the server, for every byte in the password buffer starting with the position pointed to by IbPassword, the client SHOULD first swap the four high bits with the four low bits and then do a bit-XOR with 0xA5 (10100101). After reading a submitted password, for every byte in the password buffer starting with the position pointed to by IbPassword, the server SHOULD first do a bit-XOR with 0xA5 (10100101) and then swap the four high bits with the four low bits.

2.2.6.4 PRELOGIN

Stream Name:

```
PRELOGIN
```

Stream Function:

A message sent by the client to set up context for login. The server responds to a client PRELOGIN message with a message of packet header type 0x04 and the packet data containing a PRELOGIN structure.

This message stream is also used to wrap SSL handshake payload, if encryption is needed. In this scenario, where PRELOGIN message is transporting the SSL handshake payload, the packet data is simply the raw bytes of the SSL handshake payload.

Stream Comments:

- Packet header type 0x12.
- A sample PRELOGIN message is in section 4.1.

Stream-Specific Rules:

```
UL_VERSION         =   ULONG          ; version of the sender

US_SUBBUILD        =   USHORT         ; sub-build number of the sender

B_FENCRYPTION      =   BYTE
B_INSTVALIDITY     =   *BYTE %x00     ; name of the instance
                                      ; of the database server that supports SQL
                                      ; or just %x00
UL_THREADID        =   ULONG          ; client application thread id
                                      ; used for debugging purposes
B_MARS             =   BYTE           ; sender requests MARS support
GUID_CONNID        =   16BYTE         ; client application trace id
                                      ; used for debugging purposes
ACTIVITYID         =   20BYTE         ; client application activity id
                                      ; used for debugging purposes
B_FEDAUTHREQUIRED  =   BYTE           ; authentication library requirement of the sender
                                      ; when using Integrated Authentication identity
NONCE              =   32BYTE         ; nonce to be encrypted by using session key from
                                      ; federated authentication handshake
```


Copyright © 2014 Microsoft Corporation.

```
TERMINATOR        =   %xFF          ; signals end of PRELOGIN message

PL_OPTION_DATA    =   *BYTE         ; actual data for the option
PL_OFFSET         =   USHORT        ; big endian
PL_OPTION_LENGTH  =   USHORT        ; big endian
PL_OPTION_TOKEN   =   BYTE          ; token value representing the option

PRELOGIN_OPTION   =   (PL_OPTION_TOKEN
                      PL_OFFSET
                      PL_OPTION_LENGTH)
                      /
                      TERMINATOR

SSL_PAYLOAD       =   *BYTE          ; SSL handshake raw payload
```

Stream Definition:

```
PRELOGIN          =   (*PRELOGIN_OPTION
                      *PL_OPTION_DATA)
                      /
                      SSL_PAYLOAD
```

PL_OPTION_TOKEN is described in the following table.

PL_OPTION_TOKEN	Value	Description
VERSION	0x00	`PL_OPTION_DATA = UL_VERSION US_SUBBUILD` UL_VERSION is represented in network byte order (big-endian). The server SHOULD use the VERSION sent by the client to the server. The client SHOULD use the version returned from the server to determine which features are enabled or disabled. The client SHOULD do this only if it is known that this feature is supported by that version of the database.<9>
ENCRYPTION	0x01	`PL_OPTION_DATA = B_FENCRYPTION`
INSTOPT	0x02	`PL_OPTION_DATA = B_INSTVALIDITY`
THREADID	0x03	`PL_OPTION_DATA = UL_THREADID` This value SHOULD be empty when being sent from the server to the client.
MARS	0x04	`PL_OPTION_DATA = B_MARS` ▪ 0x00 = Off ▪ 0x01 = On

Copyright © 2014 Microsoft Corporation.

PL_OPTION_TOKEN	Value	Description
TRACEID	0x05	PL_OPTION_DATA = GUID_CONNID ACTIVITYID
FEDAUTHREQUIRED	0x06	PL_OPTION_DATA = B_FEDAUTHREQUIRED
NONCEOPT	0x07	PL_OPTION_DATA = NONCE The client MUST send this option if it expects to be able to use federated authentication with Live ID Compact Token to authenticate to the server on this connection. If the server understands the NONCEOPT option and the client sends the option, the server MUST respond with its own NONCEOPT.
TERMINATOR	0xFF	Termination token.

Notes

- PL_OPTION_TOKEN VERSION is a required token, and it MUST be the first token sent as part of PRELOGIN. If this is not the case, the connection is closed by the server.
- TERMINATOR is a required token, and it MUST be the last token of PRELOGIN_OPTION. TERMINATOR does not include length and bits specifying offset.
- If encryption is agreed upon during pre-login, SSL negotiation between client and server happens immediately after the PRELOGIN packet. Then login proceeds. For more information, see section 3.3.5.1.
- A PRELOGIN message wrapping the SSL_PAYLOAD will occur only after the initial PRELOGIN message containing the PRELOGIN_OPTION and PL_OPTION_DATA information.

Encryption

During the pre-login handshake, the client and the server will negotiate the wire encryption to be used. The possible encryption option values are as follows.

Setting	Value	Description
ENCRYPT_OFF	0x00	Encryption is available but off.
ENCRYPT_ON	0x01	Encryption is available and on.
ENCRYPT_NOT_SUP	0x02	Encryption is not available.
ENCRYPT_REQ	0x03	Encryption is required.

The client sends the server the value ENCRYPT_OFF, ENCRYPT_NOT_SUP, or ENCRYPT_ON. Depending upon whether the server has encryption available and enabled, the server will respond with an ENCRYPTION value in the response according to the following table.

Client	Server ENCRYPT_OFF	Server ENCRYPT_ON	Server ENCRYPT_NOT_SUP
ENCRYPT_OFF	ENCRYPT_OFF	ENCRYPT_REQ	ENCRYPT_NOT_SUP

Copyright © 2014 Microsoft Corporation.

Client	Server ENCRYPT_OFF	Server ENCRYPT_ON	Server ENCRYPT_NOT_SUP
ENCRYPT_ON	ENCRYPT_ON	ENCRYPT_ON	ENCRYPT_NOT_SUP (connection terminated)
ENCRYPT_NOT_SUP	ENCRYPT_NOT_SUP	ENCRYPT_REQ (connection terminated)	ENCRYPT_NOT_SUP

Assuming that the client is capable of encryption, the server will require the client to behave in the following manner.

Client	Value returned from server is ENCRYPT_OFF	Value returned from server is ENCRYPT_ON	Value returned from server is ENCRYPT_REQ	Value returned from server is ENCRYPT_NOT_SUP
ENCRYPT_OFF	Encrypt login packet only	Encrypt entire connection	Encrypt entire connection	No encryption
ENCRYPT_ON	Error (connection terminated)	Encrypt entire connection	Encrypt entire connection	Error (connection terminated)

If the client and server negotiate to enable encryption, an SSL handshake will take place immediately after the initial PRELOGIN/table response message exchange. The SSL payloads MUST be transported as data in TDS buffers with the message type set to 0x12 in the packet header. For example:

```
0x 12 01 00 4e 00 00 00 00// Buffer Header
0x 16 03 01 00 &// SSL payload
```

This applies to SSL traffic. The client sends the SSL handshake payloads as data in a PRELOGIN message. For TDS versions earlier than TDS 7.2, the server SHOULD send the SSL handshake payloads as data in a table response message (0x04). For TDS 7.2, 7.3, and 7.4, the server SHOULD send the SSL handshake payloads as data in a PRELOGIN message. Upon successful completion of the SSL handshake, the client will proceed to send the LOGIN7 stream to the server to initiate authentication.

Instance Name

If available, the client SHOULD send the server the name of the instance to which it is connecting as a NULL-terminated multi-byte character set (MBCS) string in the INSTOPT option. If the string is empty or is case-insensitively equal, by using the server's locale for comparison to either the server's instance name or "MSSQLServer", the server SHOULD<10> return an INSTOPT containing a byte with the value 0 to indicate that the client's INSTOPT matches the server's instance. Otherwise, the server SHOULD return an INSTOPT containing a byte with the value of 1. The client SHOULD use the INSTOPT value from the server's PRELOGIN response for verification purposes and SHOULD terminate the connection if the INSTOPT option has the value 1.

Authentication Requirement

When the client wants to use either SSPI or federated authentication to determine the authentication mechanism but does not necessarily have a requirement as to which library to use, the client can use the FEDAUTHREQUIRED option to negotiate whether the server has a requirement for a given authentication mechanism. If the client's PRELOGIN request message contains the

Copyright © 2014 Microsoft Corporation.

FEDAUTHREQUIRED option, the client MUST specify 0x01 as the B_FEDAUTHREQUIRED value. If the server supports the FEDAUTHREQUIRED option, the server MUST respond with a FEDAUTHREQUIRED option that has either 0x00 or 0x01 as the B_FEDAUTHREQUIRED value. For the choice between SSPI and federated authentication, a value of 0x00 indicates that the server does not require federated authentication as the authentication mechanism, and a value of 0x01 indicates that the server requires federated authentication as the authentication mechanism. However, this mechanism is used only for capability negotiation when choosing between SSPI and federated authentication and does not necessarily bind the actual authentication mechanism that is used.

2.2.6.5 RPC Request

Stream Name:

```
RPCRequest
```

Stream Function:

Request to execute an RPC.

Stream Comments:

- Packet header type 0x03.
- To execute an RPC on the server, the client sends an RPCRequest data stream to the server. This is a binary stream that contains the RPC Name (or ProcID), Options, and Parameters. Each RPC MUST be contained within a separate message and not mixed with other SQL statements.
- A sample RPCRequest message is in section 4.8.

Stream-Specific Rules:

```
ProcID           =   USHORT
ProcIDSwitch     =   %xFF %xFF
ProcName         =   US_VARCHAR
NameLenProcID    =   ProcName
                     /
                     (ProcIDSwitch ProcID)

fWithRecomp      =   BIT
fNoMetaData      =   BIT
fReuseMetaData   =   BIT
OptionFlags      =   fWithRecomp
                     fNoMetaData
                     fReuseMetaData
                     13FRESERVEDBIT

fByRefValue      =   BIT
fDefaultValue    =   BIT
StatusFlags      =   fByRefValue
                     fDefaultValue
                     6FRESERVEDBIT

ParamMetaData    =   B_VARCHAR
                     StatusFlags
                     (TYPE_INFO / TVP_TYPE_INFO)    ; (TVP_TYPE_INFO introduced in TDS 7.3)
```


Copyright © 2014 Microsoft Corporation.

```
ParamLenData     =   TYPE_VARBYTE

ParameterData    =   ParamMetaData
                     ParamLenData;

BatchFlag        =   %x80 / %xFF   ; (Changed to %xFF in TDS 7.2)
NoExecFlag       =   %xFE          ; (introduced in TDS 7.2)

RPCReqBatch      =   NameLenProcID
                     OptionFlags
                     *ParameterData
```

The length for the instance value of UDTs is specified as a ULONGLONG. Also note that ParameterData is repeated once for each parameter in the request.

A StatusFlags of fDefaultValue bit MUST be zero for TVP_TYPE_INFO.

fByRefValue MUST be zero for TVP_TYPE_INFO.

Stream Definition:

```
RPCRequest       =   ALL_HEADERS
                     RPCReqBatch
                     *((BatchFlag / NoExecFlag) RPCReqBatch)
                     [BatchFlag / NoExecFlag]
```

Note that RpcReqBatch is repeated once for each RPC in the batch.

Stream Parameter Details:

Parameter	Description
ProcID	The number identifying the special stored procedure to be executed. The valid numbers with associated special stored procedure are as follows: ▪ Sp_Cursor = 1 ▪ Sp_CursorOpen = 2 ▪ Sp_CursorPrepare = 3 ▪ Sp_CursorExecute = 4 ▪ Sp_CursorPrepExec = 5 ▪ Sp_CursorUnprepare = 6 ▪ Sp_CursorFetch = 7 ▪ Sp_CursorOption = 8 ▪ Sp_CursorClose = 9

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	▪ Sp_ExecuteSql = 10 ▪ Sp_Prepare = 11 ▪ Sp_Execute = 12 ▪ Sp_PrepExec = 13 ▪ Sp_PrepExecRpc = 14 ▪ Sp_Unprepare = 15
ProcIDSwitch	ProcIDSwitch can occur as part of NameLenProcID (see below).
ProcName	The procedure name length (within US_VARCHAR), which MUST be no more than 1046 bytes.
NameLenProcID	If the first USHORT contains 0xFFFF the following USHORT contains the PROCID. Otherwise, NameLenProcID contains the parameter name length and parameter name.
OptionFlags	Bit flags in least significant bit order: ▪ fWithRecomp: 1 if RPC is sent with the "with recompile" option. ▪ fNoMetaData: 1 if the client has already cached the metadata for the result set from previous calls to the same RPC, and wants the server to avoid sending metadata by using NoMetaData see COLMETADATA (section 2.2.7.4). ▪ fReuseMetaData: 1 if the metadata has not changed from the previous call and the server SHOULD reuse its cached metadata (the metadata MUST still be sent).
StatusFlags	Bit flags in least significant bit order: ▪ fByRefValue: 1 if the parameter is passed by reference (OUTPUT parameter) OR 0 if parameter is passed by value. ▪ fDefaultValue: 1 if the parameter being passed is to be the default value.
ParameterData	The parameter name length and parameter name (within B_VARCHAR), the TYPE_INFO of the RPC data and the type-dependent data for the RPC (within TYPE_VARBYTE).
BatchFlag	Distinguishes the start of the next RPC from another parameter within the current RPC. If the version of TDS in use supports these flags, either the BatchFlag element or the NoExecFlag element MUST be present when another RPC request is in the current batch. BatchFlag SHOULD NOT be sent after the last RPCReqBatch. If BatchFlag is received after the last RPCReqBatch is received, the server MUST ignore it.
NoExecFlag	Indicates that the preceding RPC will not be executed. If this separator is found, the previous RPC will not be executed. Instead, an error message will be returned, followed by the DONEPROC marking that the RPC in the batch has finished, and then execution proceeds to the next RPC in the batch. The tabular data set returned will be very similar to what happens if the RPC does not exist—never execute the RC, just return an error message, followed by DONEPROC, and then execute the next RPC.

2.2.6.6 SQLBatch

Stream Name:

Copyright © 2014 Microsoft Corporation.

```
SQLBatch
```

Stream Function:

Describes the format of the SQL Batch message.

Stream Comments:

- Packet header type 0x01.
- A sample SQLBatch message is in section 4.6.

Stream-Specific Rules:

```
SQLText          =   UNICODESTREAM
```

Stream Definition:

```
SQLBatch         =   ALL_HEADERS
                     SQLText
```

The Unicode stream contains the text of the batch. The following is an example of a valid value for SQLText as follows.

```
Select author_id from Authors
```

2.2.6.7 SSPI Message

Stream Name:

```
SSPI
```

Stream Function:

A request to supply data for Security Support Provider Interface (SSPI) security. Note that SSPI uses the Simple and Protected GSS-API Negotiation Mechanism (**SPNEGO**) [RFC4178] negotiation.

Stream Comments:

- Packet header type 0x11.
- The initial SSPI data block (the initial SPNEGO security token) is sent from the client to the server in the LOGIN7 message. The server MUST respond with an SSPI token that is the SPNEGO security token response from the server. The client MUST respond with another SSPI message, after calling the SPNEGO interface with the server's response.

Copyright © 2014 Microsoft Corporation.

- This continues until completion or an error.
- The server completes the SSPI validation and returns the last SPNEGO security token as an SSPI token within a LOGINACK token.
- A sample SSPI message is in section 4.11.

Stream-Specific Rules:

```
SSPIData          =    BYTESTREAM
```

Stream Definition:

```
SSPI              =    SSPIData
```

Stream Parameter Details

Parameter	Description
SSPIData	The SSPIData length and SSPIData data using US_VARCHAR format.

2.2.6.8 Transaction Manager Request

Stream Name:

```
TransMgrReq
```

Stream Function:

Query and control operations pertaining to the lifecycle and state of local and distributed transaction objects. Note that distributed transaction operations are coordinated through a **Distributed Transaction Coordinator (DTC)** implemented to the DTC Interface Specification. For more information about DTC, see [MSDN-DTC].

Stream Comments:

- Packet header type 0x0E.
- A sample Transaction Manager Request message is given in section 4.13.

Stream-Specific Rules:

```
RequestType       =    USHORT
```

Stream Definition:

Copyright © 2014 Microsoft Corporation.

```
TransMgrReq      =   ALL_Headers
                     RequestType
                     [RequestPayload]
```

RequestPayload details are as specified in the following table.

Stream Parameter Details

Parameter	Description
RequestType	The types of transaction manager operations desired by the client are specified as follows. If an unknown Type is specified, the message receiver SHOULD disconnect the connection. ▪ 0 = TM_GET_DTC_ADDRESS. Returns DTC network address as a result set with a single-column, single-row binary value. ▪ 1 = TM_PROPAGATE_XACT. Imports DTC transaction into the server and returns a local transaction descriptor as a varbinary result set. ▪ 5 = TM_BEGIN_XACT. Begins a transaction and returns the descriptor in an ENVCHANGE type 8. ▪ 6 = TM_PROMOTE_XACT. Converts an active local transaction into a distributed transaction and returns an opaque buffer in an ENVCHANGE type 15. ▪ 7 = TM_COMMIT_XACT. Commits a transaction. Depending on the payload of the request, it can additionally request that another local transaction be started. ▪ 8 = TM_ROLLBACK_XACT. Rolls back a transaction. Depending on the payload of the request, it can indicate that after the rollback, a local transaction is to be started. ▪ 9 = TM_SAVE_XACT. Sets a savepoint within the active transaction. This request MUST specify a nonempty name for the savepoint. The request types 5 - 9 were introduced in TDS 7.2.

RequestPayload	▪ For RequestType TM_GET_DTC_ADDRESS: The RequestPayload SHOULD be a zero-length US_VARBYTE.
	`RequestPayload  =  US_VARBYTE`
	▪ For RequestType TM_PROPAGATE_XACT: Data contains an opaque buffer used by the server to enlist in a DTC transaction (for more information, see [MSDN-ITrans]).
	`RequestPayload  =  US_VARBYTE`
	▪ For RequestType TM_BEGIN_XACT:
	`ISOLATION_LEVEL  =  BYTE`
	`BEGIN_XACT_NAME  =  B_VARBYTE`

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	RequestPayload = ISOLATION_LEVEL BEGIN_XACT_NAME This request begins a new transaction, or increments trancount if already in a transaction. If BEGIN_XACT_NAME is nonempty, a transaction is started with the specified name. See the definition for isolation level at the end of this table. ▪ For RequestType TM_PROMOTE_XACT – No payload. This message promotes the transaction of the current request (specified in the Transaction Descriptor header). The current transaction MUST be part of the specified header. Note that TM_PROMOTE_XACT is supported only for transactions initiated via TM_BEGIN_XACT, or via piggy back operation on TM_COMMIT/TM_ROLLBACK. An error is returned if TM_PROMOTE_XACT is invoked for a TSQL initiated transaction. ▪ For RequestType TM_COMMIT_XACT: fBeginXact = BIT XACT_FLAGS = fBeginXact 7FRESERVEDBIT ISOLATION_LEVEL = BYTE XACT_NAME = B_VARBYTE BEGIN_XACT_NAME = B_VARBYTE RequestPayload = XACT_NAME XACT_FLAGS [ISOLATION_LEVEL BEGIN_XACT_NAME] Without additional flags specified, this command is semantically equivalent to issuing a TSQL COMMIT statement. The flags in XACT_FLAGS are represented in least significant bit order. If fBeginXact is 1, then a new local transaction is started after the commit operation is done. If fBeginXact is 1, then ISOLATION_LEVEL can specify the isolation level to use to start the new transaction, according to the definition at the end of this table. If fBeginXact is 0, then ISOLATION_LEVEL SHOULD NOT be present. Specifying ISOLATION_LEVEL allows the isolation level to remain in effect

Copyright © 2014 Microsoft Corporation.

Parameter	Description

	for the session, once the xact ends.
	
	If fBeginXact is 0, BEGIN_XACT_NAME SHOULD NOT be present. If fBeginXact is 1, BEGIN_XACT_NAME is nonempty.
	
	If fBeginXact is 1, a transaction MUST be started with the specified name.
	
	See the definition for isolation level at the end of this table.
	
	▪ For RequestType TM_ROLLBACK_XACT:
	
	`fBeginXact = BIT`
	
	`XACT_FLAGS = fBeginXact`
	`7FRESERVEDBIT`
	
	`ISOLATION_LEVEL = BYTE`
	
	`XACT_NAME = B_VARBYTE`
	`BEGIN_XACT_NAME = B_VARBYTE`
	
	`RequestPayload = XACT_NAME`
	`XACT_FLAGS`
	`[ISOLATION_LEVEL`
	`BEGIN_XACT_NAME]`
	
	The flags in XACT_FLAGS are represented in least significant bit order.
	
	If XACT_NAME is nonempty, this request rolls back the named transaction. This implies that if XACT_NAME specifies a savepoint name, the rollback only goes back until the specified savepoint.
	
	Without additional flags specified, this command is semantically equivalent to issuing a TSQL ROLLBACK statement under the current transaction.
	
	If fBeginXact is 1, then a new local transaction is started after the commit operation is done.
	
	If fBeginXact is 1, then ISOLATION_LEVEL can specify the isolation level to use to start the new transaction, according to the definition at the end of this table. If fBeginXact is 0, then ISOLATION_LEVELSHOULD NOT be present.
	
	Specifying ISOLATION_LEVEL allows the isolation level to remain in effect for the session, once the xact ends.
	
	If fBeginXact is 0, BEGIN_XACT_NAME SHOULD NOT be present. If fBeginXact is 1, BEGIN_XACT_NAME MAY be nonempty.
	
	If fBeginXact is 1, a transaction MUST be started with the specified name.
	
	If fBeginXact is 1, and the ROLLBACK only rolled back to a savepoint, the

Copyright © 2014 Microsoft Corporation.

Parameter	Description

	Begin_Xact operation is ignored and trancount remains unchanged.
	
	See the definition for isolation level at the end of this table.
	
	▪ For RequestType TM_SAVE_XACT:
	
	`XACT_SAVEPOINT_NAME = B_VARBYTE`
	
	`RequestPayload = XACT_SAVEPOINT_NAME`
	
	A nonempty name MUST be specified as part of this request. Otherwise, an error is raised.

ISOLATION_LEVEL MUST have one of the following values.

Value	Description
0x00	No isolation level change requested. Use current.
0x01	Read Uncommitted.
0x02	Read Committed.
0x03	Repeatable Read.
0x04	Serializable.
0x05	Snapshot.

2.2.7 Packet Data Token Stream Definition

This section describes the various tokens supported in a token-based packet data stream, as described in section 2.2.4.2. The corresponding message types that use token-based packet data streams are identified in the table in section 2.2.4.

2.2.7.1 ALTMETADATA

Token Stream Name:

```
ALTMETADATA
```

Token Stream Function:

Describes the data type, length, and name of column data that result from a SQL statement that generates totals.

Token Stream Comments:

The token value is 0x88.

Copyright © 2014 Microsoft Corporation.

This token is used to tell the client the data type and length of the column data. It describes the format of the data found in an ALTROW data stream. The ALTMETADATA and corresponding ALTROW MUST be in the same result set.

All ALTMETADATA data streams are grouped.

A preceding COLMETADATA MUST exist before an ALTMETADATA token. There might be COLINFO and TABNAME streams between COLMETADATA and ALTMETADATA.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Count            =   USHORT
Id               =   USHORT
ByCols           =   UCHAR

Op               =   BYTE
Operand          =   USHORT
UserType         =   USHORT/ULONG; (changed to ULONG in TDS 7.2)


fNullable        =   BIT
fCaseSen         =   BIT
usUpdateable     =   2BIT        ; 0 = ReadOnly
                                 ; 1 = Read/Write
                                 ; 2 = Unused
fIdentity        =   BIT
fComputed        =   BIT         ; (introduced in TDS 7.2)
usReservedODBC   =   2BIT
fFixedLenCLRType =   BIT         ; (introduced in TDS 7.2)
usReserved       =   7BIT

Flags            =   fNullable
                     fCaseSen
                     usUpdateable
                     fIdentity
                     (FRESERVEDBIT / fComputed)
                     usReservedODBC
                     (FRESERVEDBIT / fFixedLenCLRType)
                     usReserved

NumParts         =   BYTE                ; (introduced in TDS 7.2)
PartName         =   US_VARCHAR          ; (introduced in TDS 7.2)

TableName        =   US_VARCHAR          ; (removed in TDS 7.2)
                     /
                     (NumParts
                     1*PartName)         ; (introduced in TDS 7.2)
ColName          =   B_VARCHAR
ColNum           =   USHORT

ComputeData      =   Op
                     Operand
                     UserType
                     Flags
                     TYPE_INFO
                     [TableName]
                     ColName
```


Copyright © 2014 Microsoft Corporation.

The **TableName** field is specified only if text, ntext, or image columns are included in the result set.

Token Stream Definition:

```
ALTMETADATA      =   TokenType
                     Count
                     Id
                     ByCols
                     *<ByCols>ColNum
                     1* ComputeData
```

Token Stream Parameter Details:

Parameter	Description
TokenType	ALTMETADATA_TOKEN
Count	The count of columns (number of aggregate operators) in the token stream.
Id	The Id of the SQL statement to which the total column formats apply. Each ALTMETADATA token MUST have its own unique Id in the same result set. This Id lets the client correctly interpret later ALTROW data streams.
ByCols	The number of grouping columns in the SQL statement that generates totals. For example, the SQL clause *compute count(sales) by year, month, division, department* has four grouping columns.

Op	The type of aggregate operator.
	`AOPSTDEV = %x30 ; Standard deviation (STDEV)`
	`AOPSTDEVP = %x31 ; Standard deviation of the population (STDEVP)`
	`AOPVAR = %x32 ; Variance (VAR)`
	`AOPVARP = %x33 ; Variance of population (VARP)`
	`AOPCNT = %x4B ; Count of rows (COUNT)`
	`AOPSUM = %x4D ; Sum of the values in the rows (SUM)`
	`AOPAVG = %x4F ; Average of the values in the rows (AVG)`
	`AOPMIN = %x51 ; Minimum value of the rows (MIN)`
	`AOPMAX = %x52 ; Maximum value of the rows (MAX)`

Operand	The column number, starting from 1, in the result set that is the operand to the aggregate operator.
UserType	The user typeID of the data type of the column. The value will be 0x0000 with the exceptions of TIMESTAMP (0x0050) and alias types (greater than 0x00FF).
Flags	These bit flags are described in least significant bit order. With the exception of **fNullable**, all of these bit flags SHOULD be set to zero. For a description of each bit flag, see section 2.2.7.4: ▪ fCaseSens ▪ fNullable is a bit flag, 1 if the column is nullable.

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	▪ usUpdateable ▪ fIdentity ▪ fComputed ▪ usReservedODBC ▪ fFixedLenCLRType
TableName	See section 2.2.7.4 for a description of TableName. This field SHOULD never be sent because SQL statements that generate totals exclude NTEXT/TEXT/IMAGE.
ColName	The column name. Contains the column name length and column name.
ColNum	USHORT specifying the column number as it appears in the COMPUTE clause. ColNum appears ByCols times.

2.2.7.2 ALTROW

Token Stream Name:

```
ALTROW
```

Token Stream Function:

Used to send a complete row of total data, where the data format is provided by the ALTMETADATA token.

Token Stream Comments:

- The token value is 0xD3.
- The ALTROW token is similar to the ROW_TOKEN, but also contains an Id field. This Id matches an Id given in ALTMETADATA (one Id for each SQL statement). This provides the mechanism for matching row data with correct SQL statements.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Id               =   USHORT

Data             =   TYPE_VARBYTE

ComputeData      =   Data
```

Token Stream Definition:

```
ALTMETADATA      =   TokenType
                     Id
```


Copyright © 2014 Microsoft Corporation.

```
1*ComputeData
```

The ComputeData element is repeated Count times (where Count is specified in ALTMETADATA_TOKEN).

Token Stream Parameter Details:

Parameter	Description
TokenType	ALTROW_TOKEN
Id	The Id of the SQL statement that generates totals to which the total column formats apply. This Id lets the client correctly interpret later ALTROW *data streams.*
Data	The actual data for the column. The TYPE_INFO information describing the data type of this data is given in the preceding COLMETADATA_TOKEN, ALTMETDATA_TOKEN, or OFFSET_TOKEN.

2.2.7.3 COLINFO

Token Stream Name:

```
COLINFO
```

Token Stream Function:

Describes the column information in browse mode (described in [MSDN-BROWSE]), sp_cursoropen, and sp_cursorfetch.

Token Stream Comments

- The token value is 0xA5.
- The TABNAME token contains the actual table name associated with COLINFO.

Token Stream Specific Rules:

```
TokenType       =   BYTE
Length          =   USHORT

ColNum          =   BYTE
TableNum        =   BYTE
Status          =   BYTE
ColName         =   B_VARCHAR

ColProperty     =   ColNum
                    TableNum
                    Status
                    [ColName]
```

The ColInfo element is repeated for each column in the result set.

Copyright © 2014 Microsoft Corporation.

Token Stream Definition:

```
COLINFO          =   TokenType
                     Length
                     1*CpLProperty
```

Token Stream Parameter Details:

Parameter	Description
TokenType	COLINFO_TOKEN
Length	The actual data length, in bytes, of the ColProperty stream. The length does not include token type and length field.
ColNum	The column number in the result set.
TableNum	The number of the base table that the column was derived from. The value is 0 if the value of Status is EXPRESSION.
Status	0x4: EXPRESSION (the column was the result of an expression). 0x8: KEY (the column is part of a key for the associated table). 0x10: HIDDEN (the column was not requested, but was added because it was part of a key for the associated table). 0x20: DIFFERENT_NAME (the column name is different than the requested column name in the case of a column alias).
ColName	The base column name. This only occurs if DIFFERENT_NAME is set in Status.

2.2.7.4 COLMETADATA

Token Stream Name:

```
COLMETADATA
```

Token Stream Function:

Describes the result set for interpretation of following ROW data streams.

Token Stream Comments:

- The token value is 0x81.
- This token is used to tell the client the data type and length of the column data. It describes the format of the data found in a ROW *data stream*.
- All COLMETADATA *data streams* are grouped together.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Count            =   USHORT

UserType         =   USHORT/ULONG; (Changed to ULONG in TDS 7.2)
```


Copyright © 2014 Microsoft Corporation.

```
fNullable        =   BIT
fCaseSen         =   BIT
usUpdateable     =   2BIT        ; 0 = ReadOnly
                                 ; 1 = Read/Write
                                 ; 2 = Unused
fIdentity        =   BIT
fComputed        =   BIT         ; (introduced in TDS 7.2)
usReservedODBC   =   2BIT        ; (only exists in TDS 7.3.A and below)
fSparseColumnSet =   BIT         ; (introduced in TDS 7.3.B)
usReserved2      =   2BIT        ; (introduced in TDS 7.3.B)
fFixedLenCLRType =   BIT         ; (introduced in TDS 7.2)
usReserved       =   4BIT
fHidden          =   BIT         ; (introduced in TDS 7.2)
fKey             =   BIT         ; (introduced in TDS 7.2)
fNullableUnknown =   BIT         ; (introduced in TDS 7.2)

Flags            =  fNullable
                    fCaseSen
                    usUpdateable
                    fIdentity
                    (FRESERVEDBIT / fComputed)
                    usReservedODBC
                    (FRESERVEDBIT / fFixedLenCLRType)
                    (usReserved   / (FRESERVEDBIT fSparseColumnSet usReserved2))
                    (FRESERVEDBIT / fHidden)
                    (FRESERVEDBIT / fKey)
                    (FRESERVEDBIT / fNullableUnknown)

NumParts         =   BYTE        ; (introduced in TDS 7.2)
PartName         =   US_VARCHAR  ; (introduced in TDS 7.2)

TableName        =   NumParts
                     1*PartName

ColName          =   B_VARCHAR

ColumnData       =   UserType
                     Flags
                     TYPE_INFO
                     [TableName]
                     ColName

NoMetaData       =   %xFF %xFF
```

The TableName element is specified only if text, ntext, or image columns are included in the result set.

Token Stream Definition:

```
COLMETADATA      =   TokenType
                     NoMetaData / (1 *ColumnData)
```

Token Stream Parameter Details:

Copyright © 2014 Microsoft Corporation.

Parameter	Description
TokenType	COLMETADATA_TOKEN
Count	The count of columns (number of aggregate operators) in the token stream. In the event that the client requested no metadata to be returned (see section 2.2.6.5 for information about the OptionFlags parameter in the RPCRequest token), the value of Count will be 0xFFFF. This has the same effect on Count as a zero value (for example, no ColumnData is sent).
UserType	The user type ID of the data type of the column. The value will be 0x0000 with the exceptions of TIMESTAMP (0x0050) and alias types (greater than 0x00FF).
Flags	Bit flags in least significant bit order: ▪ fCaseSen is a bit flag. Set to 1 for string columns with binary collation and always for the XML data type. Set to 0 otherwise. ▪ fNullable is a bit flag. Its value is 1 if the column is nullable. ▪ usUpdateable is a 2-bit field. Its value is 0 if column is read-only, 1 if column is read/write and 2 if updateable is unknown. ▪ fIdentity is a bit flag. Its value is 1 if the column is an identity column. ▪ fComputed is a bit flag. Its value is 1 if the column is a COMPUTED column. ▪ usReservedODBC is a 2-bit field that is used by ODS gateways supporting the ODBC ODS gateway driver. ▪ fFixedLenCLRType is a bit flag. Its value is 1 if the column is a fixed-length **CLR UDT**. ▪ fSparseColumnSet is a bit flag. Set to 1 if the column is the special XML column for the sparse column set. For information about using column sets, see [MSDN-ColSets]. ▪ fHidden is a bit flag. Its value is 1 if the column is part of a hidden primary key created to support a T-SQL SELECT statement containing FOR BROWSE. ▪ fKey is a bit flag. Its value is 1 if the column is part of a primary key for the row and the T-SQL SELECT statement contains FOR BROWSE. ▪ fNullableUnknown is a bit flag. Its value is 1 if it is unknown whether the column might be nullable.
TableName	The fully qualified base table name for this column. Contains the table name length and table name. This exists only for text, ntext and image columns. Specifies how many parts will be returned and then repeats PartName once for each NumParts.
ColName	The column name. Contains the column name length and column name.
NoMetaData	This notifies client that no metadata will follow the COLMETADATA token. Client notifies the server that it has already cached the metadata from previous request, by setting fNoMetadata to 1 in RPC Request (section 2.2.6.5). The server SHOULD not send NoMetaData unless fNoMetadata is set to 1 in the request.

2.2.7.5 DONE

Token Stream Name:

Copyright © 2014 Microsoft Corporation.

```
DONE
```

Token Stream Function:

Indicates the completion status of a SQL statement.

Token Stream Comments

- The token value is 0xFD.
- This token is used to indicate the completion of a SQL statement. As multiple SQL statements can be sent to the server in a single SQL batch, multiple DONE tokens can be generated. In this case, all but the final DONE token will have a Status value with DONE_MORE bit set (details follow).
- A DONE token is returned for each SQL statement in the SQL batch except variable declarations.
- For execution of SQL statements within stored procedures, DONEPROC and DONEINPROC tokens are used in place of DONE tokens.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Status           =   USHORT
CurCmd           =   USHORT
DoneRowCount     =   LONG / ULONGLONG;  (Changed to ULONGLONG in TDS 7.2)
```

The type of the DoneRowCount element depends on the version of TDS.

Token Stream Definition:

```
DONE             =   TokenType
                     Status
                     CurCmd
                     DoneRowCount
```

Token Stream Parameter Details:

Parameter	Description
TokenType	DONE_TOKEN
Status	The Status field MUST be a bitwise 'OR' of the following: ▪ 0x00: DONE_FINAL. This DONE is the final DONE in the request. ▪ 0x1: DONE_MORE. This DONE message is not the final DONE message in the response. Subsequent data streams to follow. ▪ 0x2: DONE_ERROR. An error occurred on the current SQL statement. A preceding ERROR token SHOULD be sent when this bit is set. ▪ 0x4: DONE_INXACT. A transaction is in progress.<11> ▪ 0x10: DONE_COUNT. The DoneRowCount value is valid. This is used to distinguish between a valid value of 0 for DoneRowCount or just an initialized variable.

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	▪ 0x20: DONE_ATTN. The DONE message is a server acknowledgement of a client ATTENTION message. ▪ 0x100: DONE_SRVERROR. Used in place of DONE_ERROR when an error occurred on the current SQL statement, which is severe enough to require the result set, if any, to be discarded.
CurCmd	The token of the current SQL statement. The token value is provided and controlled by the application layer, which utilizes TDS. The TDS layer does not evaluate the value.
DoneRowCount	The count of rows that were affected by the SQL statement. The value of DoneRowCount is valid if the value of Status includes DONE_COUNT.<12>

2.2.7.6 DONEINPROC

Token Stream Name:

```
DONEINPROC
```

Token Stream Function:

Indicates the completion status of a SQL statement within a stored procedure.

Token Stream Comments

- The token value is 0xFF.
- A DONEINPROC token is sent for each executed SQL statement within a stored procedure.
- A DONEINPROC token MUST be followed by another DONEPROC token or a DONEINPROC token.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Status           =   USHORT
CurCmd           =   USHORT
DoneRowCount     =   LONG / ULONGLONG;  (Changed to ULONGLONG in TDS 7.2)
```

The type of the DoneRowCount element depends on the version of TDS.

Token Stream Definition:

```
DONEINPROC       =   TokenType
                     Status
                     CurCmd
                     DoneRowCount
```

Token Stream Parameter Details:

Copyright © 2014 Microsoft Corporation.

Parameter	Description
TokenType	DONEINPROC_TOKEN
Status	The Status field MUST be a bitwise 'OR' of the following: ▪ 0x1: DONE_MORE. This DONEINPROC message is not the final DONE/DONEPROC/DONEINPROC message in the response; more data streams are to follow. ▪ 0x2: DONE_ERROR. An error occurred on the current SQL statement or execution of a stored procedure was interrupted. A preceding ERROR token SHOULD be sent when this bit is set. ▪ 0x4: DONE_INXACT. A transaction is in progress.<13> ▪ 0x10: DONE_COUNT. The DoneRowCount value is valid. This is used to distinguish between a valid value of 0 for DoneRowCount or just an initialized variable. ▪ 0x100: DONE_SRVERROR. Used in place of DONE_ERROR when an error occurred on the current SQL statement that is severe enough to require the result set, if any, to be discarded.
CurCmd	The token of the current SQL statement. The token value is provided and controlled by the application layer, which utilizes TDS. The TDS layer does not evaluate the value.
DoneRowCount	The count of rows that were affected by the SQL statement. The value of DoneRowCount is valid if the value of Status includes DONE_COUNT.

2.2.7.7 DONEPROC

Token Stream Name:

```
DONEPROC
```

Token Stream Function:

Indicates the completion status of a stored procedure. This is also generated for stored procedures executed through SQL statements.

Token Stream Comments:

- The token value is 0xFE.
- A DONEPROC token is sent when all the SQL statements within a stored procedure have been executed.
- A DONEPROC token can be followed by another DONEPROC token or a DONEINPROC only if the DONE_MORE bit is set in the Status value.
- There is a separate DONEPROC token sent for each stored procedure called.

Token Stream-Specific Rules:

```
TokenType       =   BYTE
Status          =   USHORT
CurCmd          =   USHORT
```


Copyright © 2014 Microsoft Corporation.

```
DoneRowCount     =   LONG / ULONGLONG;  (Changed to ULONGLONG in TDS 7.2)
```

The type of the DoneRowCount element depends on the version of TDS.

Token Stream Definition:

```
DONEPROC         =   TokenType
                     Status
                     CurCmd
                     DoneRowCount
```

Token Stream Parameter Details:

Parameter	Description
TokenType	DONEPROC_TOKEN
Status	The Status field MUST be a bitwise 'OR' of the following: ▪ 0x00: DONE_FINAL. This DONEPROC is the final DONEPROC in the request. ▪ 0x1: DONE_MORE. This DONEPROC message is not the final DONEPROC message in the response; more data streams are to follow. ▪ 0x2: DONE_ERROR. An error occurred on the current stored procedure. A preceding ERROR token SHOULD be sent when this bit is set. ▪ 0x4: DONE_INXACT. A transaction is in progress.<14> ▪ 0x10: DONE_COUNT. The DoneRowCount value is valid. This is used to distinguish between a valid value of 0 for DoneRowCount or just an initialized variable. ▪ 0x80: DONE_RPCINBATCH. This DONEPROC message is associated with an RPC within a set of batched RPCs. This flag is not set on the last RPC in the RPC batch. ▪ 0x100: DONE_SRVERROR. Used in place of DONE_ERROR when an error occurred on the current stored procedure, which is severe enough to require the result set, if any, to be discarded.
CurCmd	The token of the SQL statement for executing stored procedures. The token value is provided and controlled by the application layer, which utilizes TDS. The TDS layer does not evaluate the value.
DoneRowCount	The count of rows that were affected by the command. The value of DoneRowCount is valid if the value of Status includes DONE_COUNT.

2.2.7.8 ENVCHANGE

Token Stream Name:

```
ENVCHANGE
```

Token Stream Function:

A notification of an environment change (for example, database, language, and so on).

Copyright © 2014 Microsoft Corporation.

Token Stream Comments:

- The token value is 0xE3.
- Includes old and new environment values.
- Type 4 (Packet size) is sent in response to a LOGIN7 message. The server MAY send a value different from the packet size requested by the client. That value MUST be greater than or equal to 512 and smaller than or equal to 32767. Both the client and the server MUST start using this value for packet size with the message following the login response message.
- Type 13 (Database Mirroring) is sent in response to a LOGIN7 message whenever connection is requested to a database that it is being served as primary in real-time log shipping. The ENVCHANGE stream reflects the name of the partner node of the database that is being log shipped.
- Type 15 (Promote Transaction) is sent in response to transaction manager requests with requests of type 6 (TM_PROMOTE_XACT).
- Type 16 (Transaction Manager Address) is sent in response to transaction manager requests with requests of type 0 (TM_GET_DTC_ADDRESS).
- Type 20 (Routing) is sent in response to a LOGIN7 message when the server wants to route the client to an alternate server. The ENVCHANGE stream returns routing information for the alternate server. If the server decides to send the Routing ENVCHANGE token, the Routing ENVCHANGE token MUST be sent after the LOGINACK token in the login response.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Length           =   USHORT

Type             =   BYTE

EnvValueData     =   Type
                     NewValue
                     [OldValue]
```

Token Stream Definition:

```
ENVCHANGE        =   TokenType
                     Length
                     EnvValueData
```

Token Stream Parameter Details

Parameter	Description
TokenType	ENVCHANGE_TOKEN
Length	The total length of the ENVCHANGE data stream (EnvValueData).
Type	The type of environment change: **Note** Types 8 to 19 were introduced in TDS 7.2. Type 20 was introduced in TDS 7.4.

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	▪ 1: Database ▪ 2: Language ▪ 3: Character set ▪ 4: Packet size ▪ 5: Unicode data sorting local id ▪ 6: Unicode data sorting comparison flags ▪ 7: SQL Collation ▪ 8: Begin Transaction (described in [MSDN-BEGIN]) ▪ 9: Commit Transaction (described in [MSDN-COMMIT]) ▪ 10: Rollback Transaction ▪ 11: Enlist DTC Transaction ▪ 12: Defect Transaction ▪ 13: Real Time Log Shipping ▪ 15: Promote Transaction ▪ 16: Transaction Manager Address<15> ▪ 17: Transaction ended ▪ 18: RESETCONNECTION/RESETCONNECTIONSKIPTRAN Completion Acknowledgement ▪ 19: Sends back name of user instance started per login request ▪ 20: Sends routing information to client

Type	Old Value	New Value
1: Database	OLDVALUE = B_VARCHAR	NEWVALUE = B_VARCHAR
2: Language	OLDVALUE = B_VARCHAR	NEWVALUE = B_VARCHAR
3: Character Set	OLDVALUE = B_VARCHAR	NEWVALUE = B_VARCHAR
4: Packet Size	OLDVALUE = B_VARCHAR	NEWVALUE = B_VARCHAR
5: Unicode data sorting local id	OLDVALUE = %x00	NEWVALUE = B_VARCHAR

Copyright © 2014 Microsoft Corporation.

Type	Old Value	New Value
6: Unicode data sorting comparison flags	OLDVALUE = %x00	NEWVALUE = B_VARCHAR
7: SQL Collation	OLDVALUE = B_VARBYTE	NEWVALUE = B_VARBYTE
8: Begin Transaction	OLDVALUE = %x00	NEWVALUE = B_VARBYTE
9: Commit Transaction	OLDVALUE = B_VARBYTE	NEWVALUE = "0x00"
10: Rollback Transaction	OLDVALUE = B_VARBYTE	NEWVALUE = %x00
11: Enlist DTC Transaction	OLDVALUE = B_VARBYTE	NEWVALUE = %x00
12: Defect Transaction	OLDVALUE = %x00	NEWVALUE = B_VARBYTE
13: Database Mirroring Partner	OLDVALUE = %x00	PARTNER_NODE = B_VARCHAR NEWVALUE = PARTNER_NODE
15: Promote Transaction	OLDVALUE = %x00	DTC_TOKEN = L_VARBYTE; NEWVALUE = DTC_TOKEN
16: Transaction Manager Address (not used)	OLDVALUE = %x00	XACT_MANAGER_ADDRESS = B_VARBYTE NEWVALUE = XACT_MANAGER_ADDRESS
17: Transaction Ended	OLDVALUE = B_VARBYTE	NEWVALUE = %x00
18: Reset Completion Acknowledgement	OLDVALUE = %x00	NEWVALUE = %x00
19: Sends back info of user instance for logins (login7) requesting so.	OLDVALUE = %x00	NEWVALUE = B_VARCHAR
20: Routing	OLDVALUE = %x00 %x00	Protocol = BYTE ProtocolProperty = USHORT AlternateServer = US_VARCHAR Protocol MUST be 0, specifying TCP-IP protocol. ProtocolProperty represents the TCP-IP port when Protocol is 0. A ProtocolProperty value of zero is not allowed when Protocol is TCP-IP. RoutingDataValue = Protocol ProtocolProperty AlternateServer RoutingDataValueLength = USHORT RoutingDataValueLength is the total length, in bytes, of the following fields: Protocol, ProtocolProperty, and AlternateServer.

Copyright © 2014 Microsoft Corporation.

Type	Old Value	New Value
		RoutingData = RoutingDataValueLength [RoutingDataValue] NEWVALUE = RoutingData

Notes

- For types 1, 2, 3, 4, 5, 6, 13, and 19, the payload is a Unicode string; the LENGTH will always reflect the number of bytes.
- ENVCHANGE types 3, 5, and 6 are only sent back to clients running TDS 7.0 or earlier.
- For Types 8, 9, 10, 11, 12 the ENVCHANGE event is returned only if the transaction lifetime is controlled by the user for example explicit transaction commands, including transactions started by SET IMPLICIT_TRANSACTIONS ON.
- For transactions started/committed under auto commit, no stream is generated.
- For operations that change only the value of @@trancount, no ENVCHANGE stream is generated.
- The payload of NEWVALUE for ENVCHANGE types 8, 11, and 17 and the payload of OLDVALUE for ENVCHANGE types 9, 10, and 12 is a ULONGLONG.
- ENVCHANGE type 11 is sent by the server to confirm that it has joined a distributed transaction as requested through a TM_PROPAGATE_XACT request from the client.
- ENVCHANGE type 12 is only sent when a batch defects from either a DTC or bound session transaction.
- LENGTH for ENVCHANGE type 15 is sent as 0x01 indicating only the length of the type token. Client drivers are responsible for reading the additional payload if type is 15.
- ENVCHANGE type 17 is sent when a batch is used that specified a descriptor for a transaction that has ended. This is only sent in the bound session case. For information about using bound sessions, see [MSDN-BOUND].
- ENVCHANGE type 18 always produces empty (0x00) old and new values. It simply acknowledges completion of execution of a RESETCONNECTION/RESETCONNECTIONSKIPTRAN request.
- ENVCHANGE type 19 is sent after LOGIN and after /RESETCONNECTION/RESETCONNECTIONSKIPTRAN when a client has requested use of user instances. It is sent prior to the LOGINACK token.
- ENVCHANGE type 20 MAY be sent back to a client running TDS 7.4 or later whether or not the fReadOnlyIntent bit is set in the preceding LOGIN7 record. Type 20 MAY be sent to a TDS client running TDS 7.1 to 7.3 but only when the fReadOnlyIntent bit is set in the preceding LOGIN7 record.

2.2.7.9 ERROR

Token Stream Name:

ERROR

Copyright © 2014 Microsoft Corporation.

Token Stream Function:

Used to send an error message to the client.

Token Stream Comments:

- The token value is 0xAA.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Length           =   USHORT
Number           =   LONG
State            =   BYTE
Class            =   BYTE
MsgText          =   US_VARCHAR
ServerName       =   B_VARCHAR
ProcName         =   B_VARCHAR
LineNumber       =   USHORT / LONG;  (Changed to LONG in TDS 7.2)
```

The type of the LineNumber element depends on the version of TDS.

Token Stream Definition:

```
ERROR            =   TokenType
                     Length
                     Number
                     State
                     Class
                     MsgText
                     ServerName
                     ProcName
                     LineNumber
```

Token Stream Parameter Details

Parameter	Description
TokenType	ERROR_TOKEN
Length	The total length of the ERROR data stream, in bytes.
Number	The error number.<16>
State	The error state, used as a modifier to the error number.
Class	The class (severity) of the error. A class of less than 10 indicates an informational message.
MsgText	The message text length and message text using US_VARCHAR format.
ServerName	The server name length and server name using B_VARCHAR format.
ProcName	The stored procedure name length and the stored procedure name using B_VARCHAR format.
LineNumber	The line number in the SQL batch or stored procedure that caused the error. Line numbers

Copyright © 2014 Microsoft Corporation.

Parameter	Description
	begin at 1; therefore, if the line number is not applicable to the message, the value of LineNumber will be 0.

Class level	Description
0-9	Informational messages that return status information or report errors that are not severe.<17>
10	Informational messages that return status information or report errors that are not severe.<18>
11-16	Errors that can be corrected by the user.
11	The given object or entity does not exist.
12	A special severity for SQL statements that do not use locking because of special options. In some cases, read operations performed by these SQL statements could result in inconsistent data, because locks are not taken to guarantee consistency.
13	Transaction deadlock errors.
14	Security-related errors, such as permission denied.
15	Syntax errors in the SQL statement.
16	General errors that can be corrected by the user.
17-19	Software errors that cannot be corrected by the user. These errors require system administrator action.
17	The SQL statement caused the database server to run out of resources (such as memory, locks, or disk space for the database) or to exceed some limit set by the system administrator.
18	There is a problem in the Database Engine software, but the SQL statement completes execution, and the connection to the instance of the Database Engine is maintained. System administrator action is required.
19	A non-configurable Database Engine limit has been exceeded and the current SQL batch has been terminated. Error messages with a severity level of 19 or higher stop the execution of the current SQL batch. Severity level 19 errors are rare and can be corrected only by the system administrator. Error messages with a severity level from 19 through 25 are written to the error log.
20-25	System problems have occurred. These are fatal errors, which means the Database Engine task that was executing a SQL batch is no longer running. The task records information about what occurred and then terminates. In most cases, the application connection to the instance of the Database Engine can also terminate. If this happens, depending on the problem, the application might not be able to reconnect. Error messages in this range can affect all of the processes accessing data in the same database and might indicate that a database or object is damaged. Error messages with a severity level from 19 through 25 are written to the error log.
20	Indicates that a SQL statement has encountered a problem. Because the problem has affected only the current task, it is unlikely that the database itself has been damaged.
21	Indicates that a problem has been encountered that affects all tasks in the current database, but

Copyright © 2014 Microsoft Corporation.

Class level	Description
	it is unlikely that the database itself has been damaged.
22	Indicates that the table or index specified in the message has been damaged by a software or hardware problem. Severity level 22 errors occur rarely. If one occurs, run DBCC CHECKDB to determine whether other objects in the database are also damaged. The problem might be in the buffer cache only and not on the disk itself. If so, restarting the instance of the Database Engine corrects the problem. To continue working, reconnect to the instance of the Database Engine; otherwise, use DBCC to repair the problem. In some cases, restoration of the database might be required. If restarting the instance of the Database Engine does not correct the problem, then the problem is on the disk. Sometimes destroying the object specified in the error message can solve the problem. For example, if the message reports that the instance of the Database Engine has found a row with a length of 0 in a non-clustered index, **delete** the index and rebuild it.
23	Indicates that the integrity of the entire database is in question because of a hardware or software problem. Severity level 23 errors occur rarely. If one occurs, run DBCC CHECKDB to determine the extent of the damage. The problem might be in the cache only and not on the disk itself. If so, restarting the instance of the Database Engine corrects the problem. To continue working, reconnect to the instance of the Database Engine; otherwise, use DBCC to repair the problem. In some cases, restoration of the database might be required.
24	Indicates a media failure. The system administrator might have to restore the database or resolve a hardware issue.

If an error is produced within a result set, the ERROR token is sent before the DONE token for the SQL statement, and such DONE token is sent with the error bit set.

2.2.7.10 FEATUREEXTACK

Token Stream Name:

```
FEATUREEXTACK
```

Token Stream Function:

Used to send an optional acknowledge message to the client for features defined in FeatureExt. The token stream is sent only along with the LOGINACK in a login response message.

Token Stream Comments:

- The token value is 0xAE.

Token Stream-Specific Rules:

```
TokenType            =   BYTE

FeatureId            =   BYTE
FeatureAckDataLen    =   DWORD
FeatureAckData       =   *BYTE

TERMINATOR           =   %xFF       ; signal of end of feature ack data
```


Copyright © 2014 Microsoft Corporation.

```
FeatureAckOpt        =    (FeatureId
                          FeatureAckDataLen
                          FeatureAckData)
                          /
                          TERMINATOR
```

Token Stream Definition:

```
FEATUREEXTACK     =    TokenType
                       1*FeatureAckOpt
```

Token Stream Parameter Details

Parameter	Description
TokenType	FEATUREEXTACK_TOKEN
FeatureId	The unique identifier number of a feature. Each feature MUST use the same ID number here as in FeatureExt. If the client did not send a request for a specific feature but the FeatureId is returned, the client MUST consider it as a TDS Protocol error and MUST terminate the connection. Each feature defines its own logic if it wants to use FeatureAckOpt to send information back to the client during the login response. Known FeatureId is described in the table below.
FeatureAckDataLen	The length of FeatureAckData, in bytes.
FeatureAckData	Ack data of specific feature. Each feature SHOULD define its own data format here if this token is chosen to acknowledge the feature.

The following table describes the FeatureExtAck feature option and description.

FeatureId	FeatureExtData Description
%0x00	Reserved.

%0x01
(SESSIONRECOVERY)
	Session Recovery feature. Content is defined as follows:
	`InitSessionStateData  =  SessionStateDataSet`
	`FeatureAckData  =  InitSessionStateData`
	SessionStateDataSet is described in section 2.2.7.19. The length of SessionStateDataSet is specified by the corresponding FeatureAckDataLen.
	On a recovery connection, the client sends a login request with SessionRecoveryDataToBe. The server MUST set the session state as requested by the client. If the server cannot do so, the server MUST fail the login request and terminate the connection.

%0x02 (FEDAUTH)	Whenever a login response stream is sent for a TDS connection whose login request included a FEDAUTH FeatureExt, the server login response message stream MUST include a FEATUREEXTACK token, and the FEATUREEXTACK token stream MUST include the FEDAUTH FeatureId. The format is described below based on the bFedAuthLibrary that is used in FEDAUTH FeatureExt. When the bFedAuthLibrary is Live ID Compact Token, the format is as follows:

Copyright © 2014 Microsoft Corporation.

FeatureId	FeatureExtData Description
	Nonce = 32BYTE Signature = 32BYTE FeatureAckData = Nonce Signature Nonce: The client-specified nonce in PRELOGIN. Signature: The HMAC-SHA-256 [RFC6234] of the client-specified nonce, using the session key retrieved from the federated authentication context as the shared secret. When the bFedAuthLibrary is Security Token, the format is as follows: Nonce = 32BYTE FeatureAckData = [Nonce] Nonce: The client-specified nonce in PRELOGIN. This field MUST be present if the client's PRELOGIN message included a NONCE field. Otherwise, this field MUST NOT be present.
%xFF (TERMINATOR)	This is the last option in FeatureExtAck.

2.2.7.11 INFO

Token Stream Name:

```
INFO
```

Token Stream Function:

Used to send an information message to the client.

Token Stream Comments

- The token value is 0xAB.

Token Stream-Specific Rules:

```
TokenType       =   BYTE
Length          =   USHORT
Number          =   LONG
State           =   BYTE
Class           =   BYTE
MsgText         =   US_VARCHAR
ServerName      =   B_VARCHAR
ProcName        =   B_VARCHAR
LineNumber      =   USHORT / ULONG;  (Changed to ULONG in TDS 7.2)
```

The type of the LineNumber element depends on the version of TDS.

Token Stream Definition:

Copyright © 2014 Microsoft Corporation.

```
INFO              =    TokenType
                       Length
                       Number
                       State
                       Class
                       MsgText
                       ServerName
                       ProcName
                       LineNumber
```

Token Stream Parameter Details

Parameter	Description
TokenType	INFO_TOKEN
Length	The total length of the INFO data stream, in bytes.
Number	The info number.<19>
State	The error state, used as a modifier to the info Number.
Class	The class (severity) of the error. A class of less than 10 indicates an informational message.
MsgText	The message text length and message text using US_VARCHAR format.
ServerName	The server name length and server name using B_VARCHAR format.
ProcName	The stored procedure name length and stored procedure name using B_VARCHAR format.
LineNumber	The line number in the SQL batch or stored procedure that caused the error. Line numbers begin at 1; therefore, if the line number is not applicable to the message as determined by the upper layer, the value of LineNumber will be 0.

2.2.7.12 LOGINACK

Token Stream Name:

```
LOGINACK
```

Token Stream Function:

Used to send a response to a login request (LOGIN7) to the client.

Token Stream Comments

- The token value is 0xAD.
- If a LOGINACK is not received by the client as part of the login procedure, the login to the server is unsuccessful.

Token Stream-Specific Rules:

```
TokenType         =    BYTE
```


Copyright © 2014 Microsoft Corporation.

```
Length          =   USHORT
Interface       =   BYTE
TDSVersion      =   DWORD
ProgName        =   B_VARCHAR

MajorVer        =   BYTE
MinorVer        =   BYTE
BuildNumHi      =   BYTE
BuildNumLow     =   BYTE

ProgVersion     =   MajorVer
                    MinorVer
                    BuildNumHi
                    BuildNumLow
```

Token Stream Definition:

```
LOGINACK        =   TokenType
                    Length
                    Interface
                    TDSVersion
                    ProgName
                    ProgVersion
```

Token Stream Parameter Details

Parameter	Description
TokenType	LOGINACK_TOKEN
Length	The total length, in bytes, of the following fields: Interface, TDSVersion, Progname, and ProgVersion.
Interface	The type of interface with which the server will accept client requests: 0: SQL_DFLT (server confirms that whatever is sent by the client is acceptable. If the client requested SQL_DFLT, SQL_TSQL will be used). 1: SQL_TSQL (TSQL is accepted).
TDSVersion	The TDS version being used by the server (for example, 0x74000004 for a server running Microsoft SQL Server 2014).<20>
ProgName	The name of the server.
MajorVer	The major version number (0-255).
MinorVer	The minor version number (0-255).
BuildNumHi	The high byte of the build number (0-255).
BuildNumLow	The low byte of the build number (0-255).

2.2.7.13 NBCROW

Token Stream Name:

Copyright © 2014 Microsoft Corporation.

```
NBCROW
```

Token Stream Function:

NBCROW, introduced in TDS 7.3.B, is used to send a row as defined by the COLMETADATA token to the client with null bitmap compression. Null bitmap compression is implemented by using a single bit to specify whether the column is null or not null and also by removing all null column values from the row. Removing the null column values (which can be up to 8 bytes per null instance) from the row provides the compression. The null bitmap contains one bit for each column defined in COLMETADATA. In the null bitmap, a bit value of 1 means that the column is null and therefore not present in the row, and a bit value of 0 means that the column is not null and is present in the row. The null bitmap is always rounded up to the nearest multiple of 8 bits, so there might be 1 to 7 leftover reserved bits at the end of the null bitmap in the last byte of the null bitmap. NBCROW is only used by TDS result set streams from server to client. NBCROW MUST NOT be used in BulkLoadBCP streams. NBCROW MUST NOT be used in TVP row streams.

Token Stream Comments

- The token value is 0xD2/210.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
TextPointer      =   B_VARBYTE
Timestamp        =   8BYTE
Data             =   TYPE_VARBYTE
NullBitmap       =   <NullBitmapByteCount>BYTE;    see note on NullBitmapByteCount
ColumnData       =   [TextPointer Timestamp] Data
AllColumnData    =   *ColumnData
```

ColumnData is repeated once for each non-null column of data.

NullBitmapBitCount is equal to the number of columns in COLMETADATA.

NullBitmapByteCount is equal to the smallest number of bytes needed to hold 'NullBitmapBitCount' bits.

The server can decide to send either a NBCROW token or a ROW token. For example, the server MAY choose to send a ROW token if there will be no byte savings if the result set has no **nullable column**s, or if a particular row in a result set has no null values. This implies that NBCROW and ROW tokens can be intermixed in the same result set.

When determining whether or not a specific column is null, consider all the columns from left to right ordered using a zero-based index from 0 to 65534 as they occur in the ColumnData section of the COLMETADATA token. The null bitmap indicates that a column is null using a zero bit at the following byte and bit layout:

```
Byte 1                  Byte 2                  Byte 3
----------------------- ----------------------- -----------------------
07 06 05 04 03 02 01 00 15 14 13 12 11 10 09 08 23 22 21 20 19 18 17 16
```


Copyright © 2014 Microsoft Corporation.

Hence the first byte will contain flags for columns 0 through 7, with the least significant (or rightmost) bit within the byte indicating the zeroth column and the most significant (or leftmost) bit within the byte indicating the seventh column. For example, column index 8 would be in the second byte as the least significant bit. If the null bitmap bit is set, the column is null and no null token value for the column will follow in the row. If the null bitmap bit is clear, the column is not null and the value for the column follows in the row.

Token Stream Definition:

```
NBCROW              = TokenType
                      NullBitmap
                      AllColumnData
```

Token Stream Parameter Details

Parameter	Description
TokenType	NBCROW_TOKEN (0xD2)
TextPointer	The length of the text pointer and the text pointer for Data.
Timestamp	The timestamp of a text/image column.
Data	The actual data for the column. The TYPE_INFO information describing the data type of this data is given in the preceding COLMETADATA_TOKEN.

2.2.7.14 OFFSET

Token Stream Name:

```
OFFSET                    ; (removed in TDS 7.2)
```

Token Stream Function:

Used to inform the client where in the client's SQL text buffer a particular keyword occurs.

Token Stream Comments:

- The token value is 0x78.
- The token was removed in TDS 7.2.

Token Stream-Specific Rules:

```
TokenType         =   BYTE
Identifier        =   USHORT
OffSetLen         =   USHORT
```

Token Stream Definition:

```
OFFSET            =   TokenType
```


Copyright © 2014 Microsoft Corporation.

```
                    Identifier
                    OffSetLen
```

Token Stream Parameter Details

Parameter	Description
TokenType	OFFSET_TOKEN
Identifier	The keyword to which OffSetLen refers.
OffsetLen	The offset in the SQL text buffer received by the server of the identifier. The SQL text buffer begins with an OffSetLen value of 0 (MOD 64 kilobytes if value of OffSet is larger than 64 kilobytes).

2.2.7.15 ORDER

Token Stream Name:

```
ORDER
```

Token Stream Function:

Used to inform the client by which columns the data is ordered.

Token Stream Comments

- The token value is 0xA9.
- This token is sent only in the event that an ORDER BY clause is executed.

Token Stream-Specific Rules:

```
TokenType         =   BYTE
Length            =   USHORT
ColNum            =   *USHORT
```

The ColNum element is repeated once for each column within the ORDER BY clause.

Token Stream Definition:

```
ORDER             =   TokenType
                      Length
                      ColNum
```

Token Stream Parameter Details

Copyright © 2014 Microsoft Corporation.

Parameter	Description
TokenType	ORDER_TOKEN
Length	The total length of the ORDER data stream.
ColNum	The column number in the result set.

2.2.7.16 RETURNSTATUS

Token Stream Name:

```
RETURNSTATUS
```

Token Stream Function:

Used to send the status value of an RPC to the client. The server also uses this token to send the result status value of a T-SQL EXEC query.

Token Stream Comments:

- The token value is 0x79.
- This token MUST be returned to the client when an RPC is executed by the server.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Value            =   LONG
```

Token Stream Definition:

```
RETURNSTATUS     =   TokenType
                     Value
```

Token Stream Parameter Details

Parameter	Description
TokenType	RETURNSTATUS_TOKEN
Value	The return status value determined by the remote procedure. Return status MUST NOT be NULL.

2.2.7.17 RETURNVALUE

Token Stream Name:

Copyright © 2014 Microsoft Corporation.

```
RETURNVALUE
```

Token Stream Function:

Used to send the return value of an RPC to the client. When an RPC is executed, the associated parameters might be defined as input or output (or "return") parameters. This token is used to send a description of the return parameter to the client. This token is also used to describe the value returned by a UDF when executed as an RPC.

Token Stream Comments:

- The token value is 0xAC.
- Multiple return values can exist per RPC. There is a separate RETURNVALUE token sent for each parameter returned.
- Large Object output parameters are reordered to appear at the end of the stream. First the group of small parameters is sent, followed by the group of large output parameters. There is no reordering within the groups.
- A UDF cannot have return parameters. As such, if a UDF is executed as an RPC there is exactly one RETURNVALUE token sent to the client.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
ParamName        =   B_VARCHAR
ParamOrdinal     =   USHORT
Status           =   BYTE
UserType         =   USHORT/ULONG; (Changed to ULONG in TDS 7.2)


fNullable        =   BIT
fCaseSen         =   BIT
usUpdateable     =   2BIT            ; 0 = ReadOnly
                                     ; 1 = Read/Write
                                     ; 2 = Unused

fIdentity        =   BIT
fComputed        =   BIT             ; (introduced in TDS 7.2)
usReservedODBC   =   2BIT
fFixedLenCLRType =   BIT             ; (introduced in TDS 7.2)
usReserved       =   7BIT

Flags            =   fNullable
                     fCaseSen
                     usUpdateable
                     fIdentity
                     (FRESERVEDBIT / fComputed)
                     usReservedODBC
                     (FRESERVEDBIT / fFixedLenCLRType)
                     usReserved

TypeInfo         =   TYPE_INFO
Value            =   TYPE_VARBYTE
```


Copyright © 2014 Microsoft Corporation.

Token Stream Definition:

```
RETURNVALUE      =   TokenType
                     ParamOrdinal
                     ParamName
                     Status
                     UserType
                     Flags
                     TypeInfo
                     Value
```

Token Stream Parameter Details:

Parameter	Description
TokenType	RETURNVALUE_TOKEN
ParamOrdinal	Indicates the ordinal position of the output parameter in the original RPC call. Large Object output parameters are reordered to appear at the end of the stream. First the group of small parameters is sent, followed by the group of large output parameters. There is no reordering within the groups.
ParamName	The parameter name length and parameter name (within B_VARCHAR).
Status	0x01: If ReturnValue corresponds to OUTPUT parameter of a stored procedure invocation. 0x02: If ReturnValue corresponds to return value of User Defined Function.
UserType	The user-defined data type of the column. The value will be 0x00 00 with the exceptions of TIMESTAMP (0x00 50) and alias types (> 0x00 FF).
Flags	These bit flags are described in least significant bit order. All of these bit flags SHOULD be set to zero. For a description of each bit flag, see section 2.2.7.4. ▪ fCaseSen ▪ fNullable ▪ usUpdateable ▪ fIdentity ▪ fComputed ▪ usReservedODBC ▪ fFixedLengthCLRType
TypeInfo	The TYPE_INFO for the message.
Value	The type-dependent data for the parameter (within TYPE_VARBYTE).

Copyright © 2014 Microsoft Corporation.

2.2.7.18 ROW

Token Stream Name:

```
ROW
```

Token Stream Function:

Used to send a complete row, as defined by the COLMETADATA token, to the client.

Token Stream Comments:

- The token value is 0xD1.

Token Stream-Specific Rules:

```
TokenType        =   BYTE

TextPointer      =   B_VARBYTE
Timestamp        =   8BYTE
Data             =   TYPE_VARBYTE

ColumnData       =   [TextPointer Timestamp]
                     Data

AllColumnData    =   *ColumnData
```

The ColumnData element is repeated once for each column of data.

TextPointer and Timestamp MUST NOT be specified if the instance of type text/ntext/image is a NULL instance (GEN_NULL).

Token Stream Definition:

```
ROW              =   TokenType
                     AllColumnData
```

Token Stream Parameter Details:

Parameter	Description
TokenType	ROW_TOKEN
TextPointer	The length of the text pointer and the text pointer for data.
Timestamp	The timestamp of a text/image column. This is not present if the value of data is CHARBIN_NULL or GEN_NULL.
Data	The actual data for the column. The TYPE_INFO information describing the data type of this data is given in the preceding COLMETADATA_TOKEN, ALTMETDATA_TOKEN or OFFSET_TOKEN.

Copyright © 2014 Microsoft Corporation.

2.2.7.19 SESSIONSTATE

Token Stream Name:

```
SESSIONSTATE
```

Token Stream Function:

Used to send session state data to the client. The data format defined here can also be used to send session state data for session recovery during login and login response.

Token Stream Comments:

- The token value is 0xE4.
- This token stream MUST NOT be sent if the SESSIONRECOVERY feature is not negotiated on the connection.
- When this token stream is sent, the next token MUST be DONE or DONEPROC with DONE_FINAL.
- If the SESSIONRECOVERY feature is negotiated on the connection, the server SHOULD send this token to the client to inform any session state update.

Token Stream-Specific Rules:

```
fRecoverable = BIT
Status       = fRecoverable 7FRESERVEDBIT

TokenType    =   BYTE
Length       =   DWORD
SeqNo        =   DWORD
Status       =   BYTE

StateId      =   BYTE
StateLen     =   BYTE       ; 0-%xFE
                /
                (%xFF
                 DWORD)     ; %xFF - %xFFFF


SessionStateData    =   StateId
                        StateLen
                        StateValue

SessionStateDataSet    =   1*SessionStateData
```

Token Stream Definition:

```
SESSIONSTATE     =   TokenType
                     Length
                     SeqNo
                     Status
                     SessionStateDataSet
```

Token Stream Parameter Details

Copyright © 2014 Microsoft Corporation.

Parameter	Description
TokenType	SESSIONSTATE_TOKEN
Length	The length, in bytes, of the token stream (excluding TokenType and Length).
SeqNo	The sequence number of the SESSIONSTATE token in the connection. This number, which starts at 0 and increases by one each time, can be used to track the order of SESSIONSTATE tokens sent during the course of a connection. The SeqNo applies to all StateIds in the token. If the SeqNo for any StateId reaches %xFFFFFFFF, both client and server MUST consider that the SESSIONRECOVERY feature is permanently disabled on the connection. The server SHOULD send a token with fRecoverable set to FALSE to disable SESSIONRECOVERY for this session. The client SHOULD NOT set either ResetConn bit (RESETCONNECTION or RESETCONNECTIONSKIPTRAN) on the connection once it receives any SeqNo of %xFFFFFFFF because ResetConn could reset a connection back to an initial recoverable state and SESSIONRECOVERY needs to be permanently disabled on the connection in this case. If the server does receive ResetConn after SeqNo reaches %xFFFFFFFF, it SHOULD reuse this same SeqNo to disable SESSIONRECOVERY. The client SHOULD track SeqNo for each StateId and keep the latest data for session recovery.
Status	Status of the session StateId in this token. fRecoverable: TRUE means all session StateIds in this token are recoverable. The client SHOULD track Status for each StateId and keep the latest data for session recovery. A client MUST NOT try to recover a dead connection unless fRecoverable is TRUE for all session StateIds received from server.
StateId	The identification number of the session state. %xFF is reserved.
StateLen	The length, in bytes, of the corresponding StateValue. If the length is 254 bytes or smaller, one BYTE is used to represent the field. If the length is 255 bytes or larger, %xFF followed by a DWORD is used to represent the field. If this field is 0, client SHOULD skip sending SessionStateData for the StateId during session recovery.
StateValue	The value of the session state. This can be any arbitrary data as long as the server understands it.

2.2.7.20 SSPI

Token Stream Name:

```
SSPI
```

Token Stream Function:

The SSPI token returned during the login process.

Token Stream Comments:

- The token value is 0xED.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
```


Copyright © 2014 Microsoft Corporation.

```
SSPIBuffer       =   US_VARBYTE
```

Token Stream Definition:

```
SSPI             =   TokenType
                     SSPIBuffer
```

Token Stream Parameter Details:

Parameter	Description
TokenType	SSPI_TOKEN
SSPIBuffer	The length of the SSPIBuffer and the SSPI buffer using B_VARBYTE format.

2.2.7.21 TABNAME

Token Stream Name:

```
TABNAME
```

Token Stream Function:

Used to send the table name to the client only when in browser mode or from sp_cursoropen.

Token Stream Comments:

- The token value is 0xA4.

Token Stream-Specific Rules:

```
TokenType        =   BYTE
Length           =   USHORT

NumParts         =   BYTE              ; (introduced in TDS 7.1 Revision 1)
PartName         =   US_VARCHAR        ; (introduced in TDS 7.1 Revision 1)

TableName        =   US_VARCHAR        ; (removed in TDS 7.1 Revision 1)
                     /
                     (NumParts
                     1*PartName)       ; (introduced in TDS 7.1 Revision 1)

AllTableNames    =   TableName
```

The TableName element is repeated once for each table name in the query.

Token Stream Definition:

Copyright © 2014 Microsoft Corporation.

```
TABNAME          =    TokenType
                      Length
                      AllTableNames
```

Token Stream Parameter Details

Parameter	Description
TokenType	TABNAME_TOKEN
Length	The actual data length, in bytes, of the TABNAME token stream. The length does not include token type and length field.
TableName	The name of the base table referenced in the query statement.

2.2.7.22 TVP ROW

Token Stream Name:

```
TVP ROW
```

Token Stream Function:

Used to send a complete table valued parameter (TVP) row, as defined by the TVP_COLMETADATA token from client to server.

Token Stream Comments:

- The token value is 0x01/1.

Token Stream-Specific Rules:

```
TokenType        =    BYTE
TvpColumnData    =    TYPE_VARBYTE
AllColumnData    =    *TvpColumnData
```

TvpColumnData is repeated once for each column of data with a few exceptions. For details about when certain TvpColumnData items are required to be omitted, see the Flags description of the TVP_COLMETADATA definition (see section 2.2.5.5.5.1).

Note that unlike the ROW token, TVP_ROW does not use TextPointer + TimeStamp prefix with TEXT, NTEXT and IMAGE types.

Token Stream Definition:

```
TVP ROW              =    TokenType
                          AllColumnData
```

Token Stream Parameter Details:

Copyright © 2014 Microsoft Corporation.

Parameter	Description
TokenType	TVP_ROW_TOKEN
TvpColumnData	The actual data for the TVP column. The TYPE_INFO information describing the data type of this data is given in the preceding TVP_COLMETADATA token.

Copyright © 2014 Microsoft Corporation.

3 Protocol Details

This section describes the important elements of the client software and the server software necessary to support the TDS protocol.

3.1 Common Details

As described in section 1.3, TDS is an application-level protocol that is used for the transfer of requests and responses between clients and database server systems. The protocol defines a limited set of messages through which the client can make a request to the server. The TDS server is message-oriented. Once a connection has been established between the client and server, a complete message is sent from client to server. Following this, a complete response is sent from server to client (with the possible exception of when the client aborts the request), and the server then waits for the next request. Other than this Post-Login state, the other states defined by the TDS protocol are (i) pre-authentication (Pre-Login), (ii) authentication (Login), and (iii) when the client sends an attention message (Attention). These will be expanded upon in subsequent sections.

3.1.1 Abstract Data Model

See sections 3.2.1 and 3.3.1 for the abstract data model of the client and server, respectively.

3.1.2 Timers

See section 3.2.2 for a description of the client timer used and section 3.3.2 for a description of the server timer used.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

For information about higher-layer triggered events, see section 3.2.4 for a TDS client and section 3.3.4 for a TDS server.

3.1.5 Message Processing Events and Sequencing Rules

The following series of sequence diagrams illustrate the possible messages that can be exchanged between client and server. See sections 3.2.5 and 3.3.5 for specific client and server details regarding message processing events and sequencing rules.

Copyright © 2014 Microsoft Corporation.

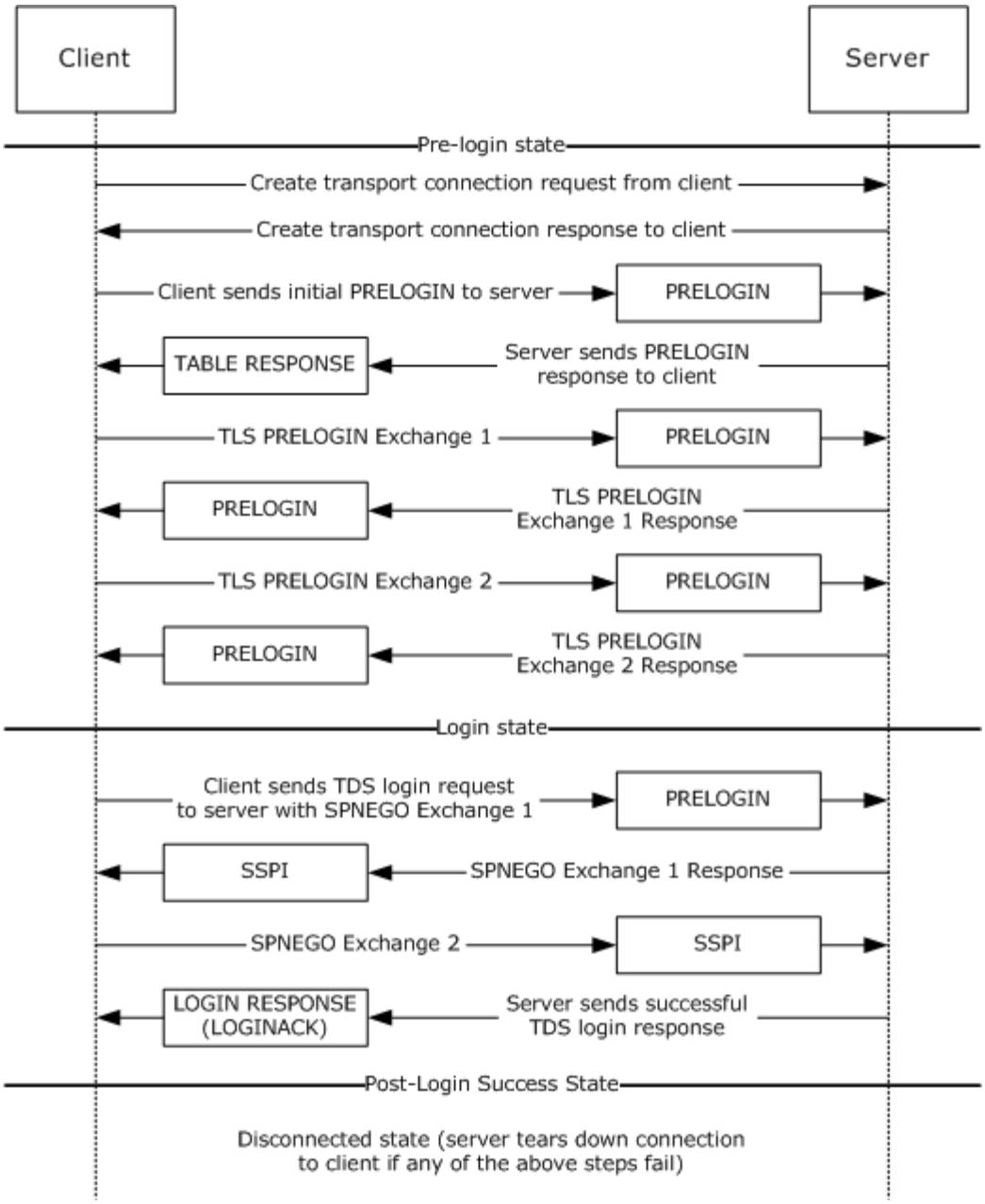

Figure 3: Pre-login to post-login sequence

Copyright © 2014 Microsoft Corporation.

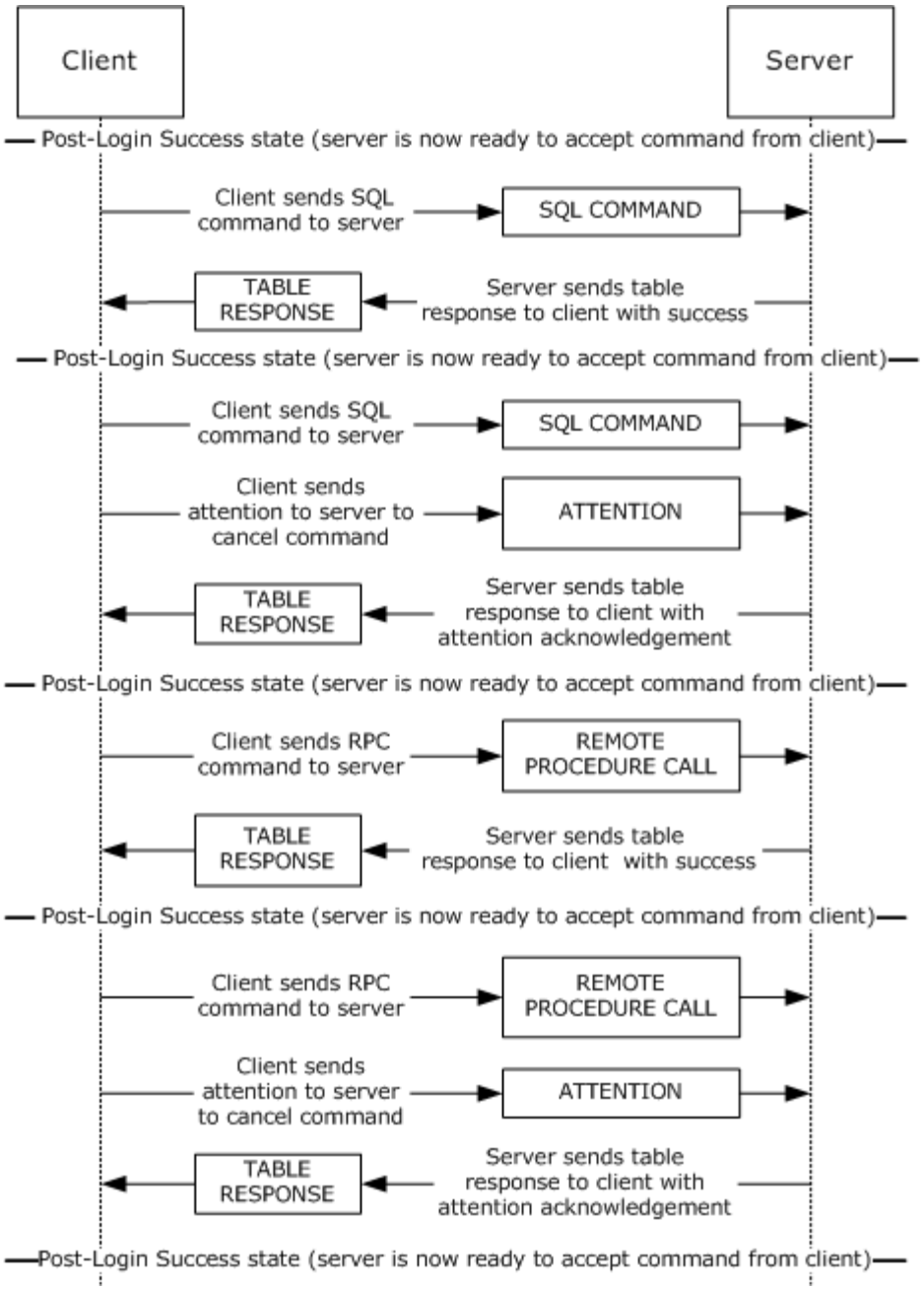

Figure 4: SQL command and RPC sequence

Copyright © 2014 Microsoft Corporation.

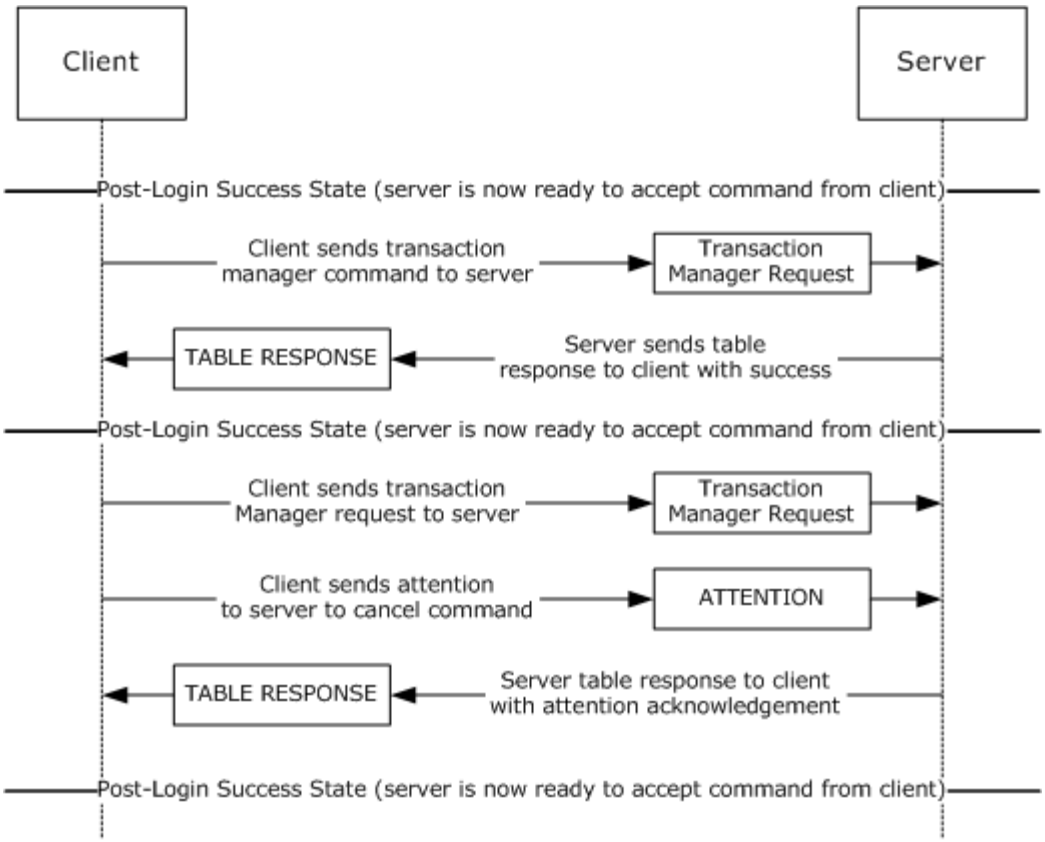

Figure 5: Transaction manager request sequence

Copyright © 2014 Microsoft Corporation.

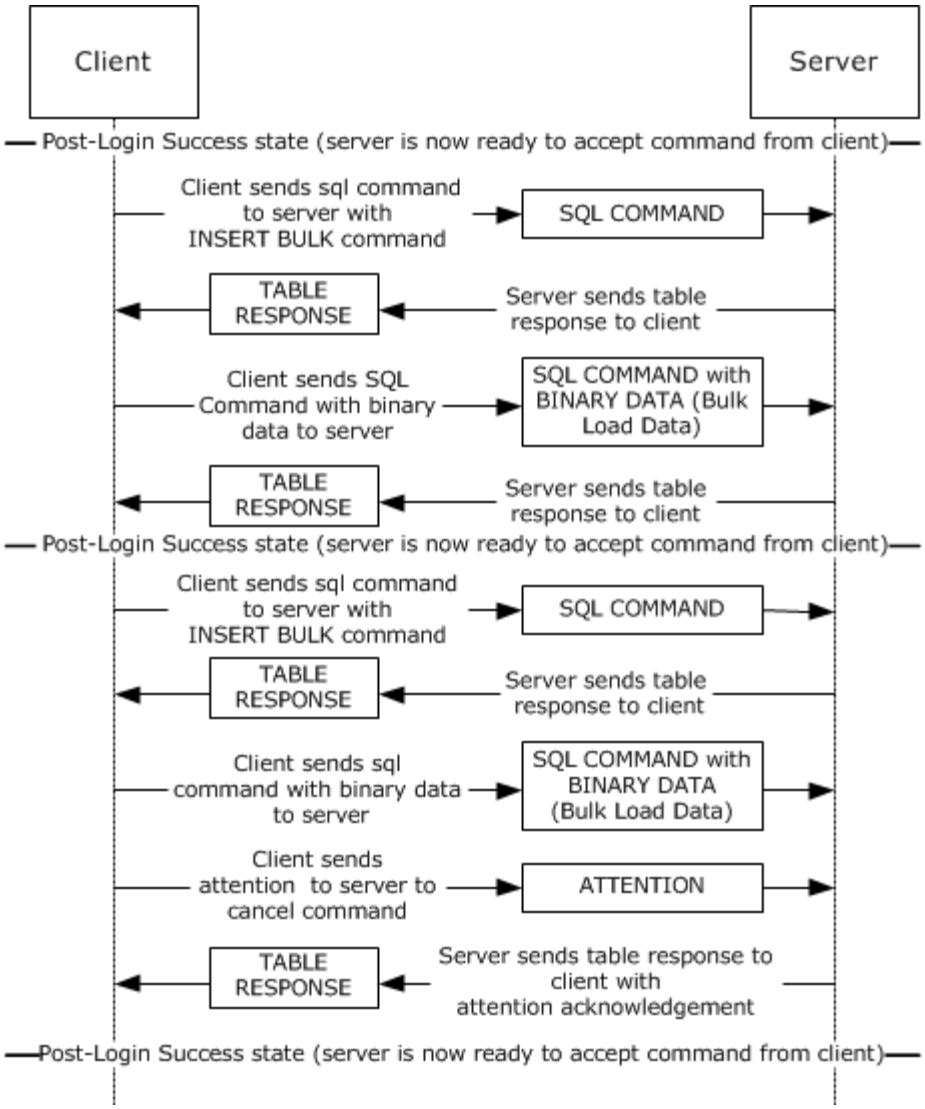

Figure 6: Bulk insert sequence

3.1.6 Timer Events

See sections 3.2.6 and 3.3.6 for the timer events of the client and server, respectively.

3.1.7 Other Local Events

A TDS session is tied to the underlying established network protocol session. As such, loss or termination of a network connection is equivalent to immediate termination of a TDS session.

See sections 3.2.7 and 3.3.7 for the other local events of the client and server, respectively.

3.2 Client Details

The following state machine diagram describes TDS on the client side.

Copyright © 2014 Microsoft Corporation.

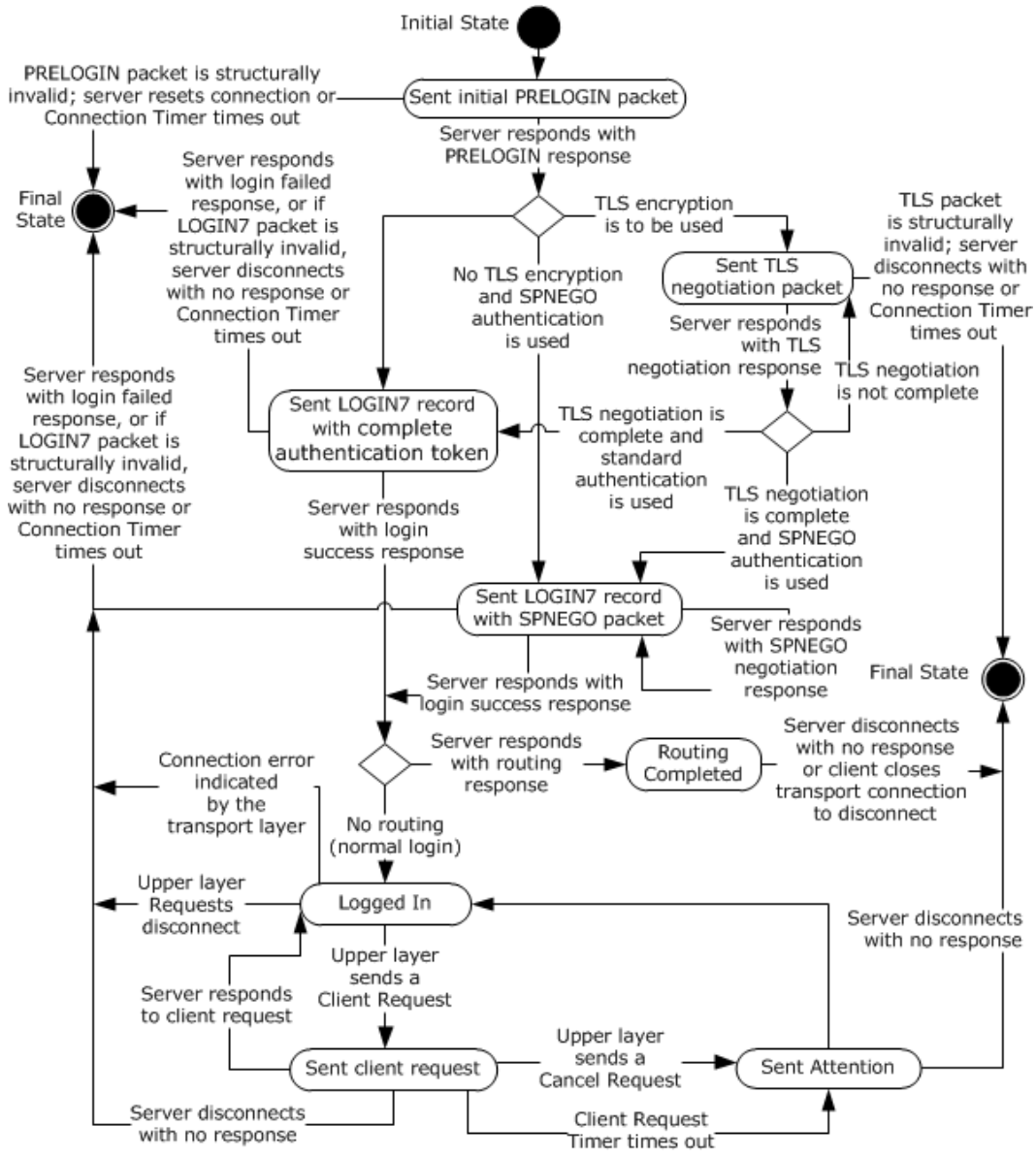

Figure 7: TDS client state machine

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

A TDS client SHOULD maintain the following states:

- Sent Initial PRELOGIN Packet State

Copyright © 2014 Microsoft Corporation.

- Sent TLS/SSL Negotiation Packet State
- Sent LOGIN7 Record with Complete Authentication Token State
- Sent LOGIN7 Record with SPNEGO Packet State
- Logged In State
- Sent Client Request State
- Sent Attention State
- Routing Completed State
- Final State

3.2.2 Timers

A TDS client SHOULD implement the following three timers:

- Connection Timer. Controls the maximum time spent during the establishment of a TDS connection. The default value SHOULD be 15 seconds. The implementation SHOULD allow the upper layer to specify a nondefault value, including an infinite value (for example, no timeout).
- Client Request Timer. Controls the maximum time spent waiting for a query response from the server for a client request sent after the connection has been established. The default value is implementation-dependent. The implementation SHOULD allow the upper layer to specify a non-default value, including an infinite value (for example, no timeout).<21>
- Cancel Timer. Controls the maximum time spent waiting for a query cancellation acknowledgement after an Attention request is sent to the server. The default value is implementation-dependent. The implementation SHOULD allow the upper layer to specify a nondefault value, including an infinite value (for example, no timeout).<22>

For all three timers, a client can implement a minimum timeout value that is as short as desired. If a TDS client implementation implements any of the timers, it MUST implement their behavior according to this specification.

A TDS client SHOULD request the transport to detect and indicate a broken connection if the transport provides such mechanism. If the transport used is TCP, it SHOULD use the TCP Keep-Alives [RFC1122] in order to detect a nonresponding server in case infinite connection timeout or infinite client request timeout is used. The default values of the TCP Keep-Alive values set by a TDS client are 30 seconds of no activity until the first keep-alive packet is sent and 1 second between when successive keep-alive packets are sent if no acknowledgement is received. The implementation SHOULD allow the upper layer to specify other TCP keep-alive values.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

A TDS client MUST support the following events from the upper layer:

- Connection Open Request to establish a new TDS connection to a TDS server.

Copyright © 2014 Microsoft Corporation.

- Client Request to send a query to a TDS server on an already established TDS connection. The Client Request is a request for one of four types of queries to be sent: SQL Command, SQL Command with Binary Data, Transaction Manager Request, or an RPC.

In addition, it SHOULD support the following event from the upper layer:

- Cancel Request to cancel a client request while waiting for a server response. For example, this enables the upper layer to cancel a long-running client request if the user/upper layer is no longer seeking the result, thus freeing up thus client and server resources. If a client implementation of the TDS protocol supports the Cancel Request event, it MUST handle it as described in this specification.

The processing and actions triggered by these events is described in the remaining parts of this section.

When a TDS client receives a Connection Open Request from the upper layer in the "Initial" state of a TDS connection, it performs the following actions:

- If the TDS client implements the Connection Timer, it MUST start the Connection Timer if the connection timeout value is not infinite.
- If there is upper-layer request MARS support, it MUST set the B_MARS byte in the PRELOGIN message to 0x01.
- It MUST send a PRELOGIN message to the server by using the underlying transport protocol.
- If the transport does not report an error, it MUST enter the "Sent Initial PRELOGIN Packet" state.

When a TDS client receives a Connection Open Request from the upper layer in any state other than the **Initial state** of a TDS connection, it MUST indicate an error to the upper layer.

When a TDS client receives a Client Request from the upper layer in the "Logged In" state, it MUST perform the following actions:

- If the TDS client implements the Query Timer, it MUST start the Client Request Timer if the client request timeout value is not infinite.
- If MARS is enabled, the client MUST keep track whether there is an outstanding active request. If this is the case, then the client MUST initiate a new SMUX session, or else an existing SMUX session MAY be used.
- Send either SQL Command, SQL Command with Binary Data, Transaction Manager Request, or a RPC message to the server. The message and its content MUST match the requested message from the Client Request. If MARS is enabled, the TDS message MUST be passed through to the SMUX layer.
- If the transport does not report an error, then enter the "Sent Client Request" state.

When a TDS client supporting the Cancel Request receives a Cancel Request from the upper layer in the "Sent Client Request" state, it MUST perform the following actions:

- If the TDS client implements the Cancel Timer, it MUST start the Cancel Timer if the Attention request timeout value is not infinite.
- Send an Attention message to the server. This indicates to the server that the client intends to abort the currently executing request. If MARS is enabled, the Attention message MUST be passed through to the SMUX layer.

Copyright © 2014 Microsoft Corporation.

- Enter the "Sent Attention" state.

3.2.5 Message Processing Events and Sequencing Rules

The processing of messages received from a TDS server depends on the message type and the current state of the TDS client. The rest of this section describes the processing and actions to take on them. The message type is determined from the TDS packet type and the token stream inside the TDS packet payload, as described in section 2.2.3.

Whenever the TDS client enters either the "Logged In" state or the "**Final State**" state, it MUST stop the Connection Timer (if implemented and running), the Client Request Timer (if implemented and running), and the Cancel Timer (if implemented and running).

Whenever a TDS client receives a **structurally invalid** TDS message, it MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

When a TDS client receives a table response (TDS packet type %x04) from the server, it MUST behave as follows, according to the state of the TDS client.

3.2.5.1 Sent Initial PRELOGIN Packet State

If the response contains a structurally valid PRELOGIN response indicating a success, the TDS client MUST take action according to the Encryption option and Authentication scheme:

- The Encryption option MUST be handled as described in section 2.2.6.4 in the PRELOGIN message description.
- If encryption was negotiated, the TDS client MUST initiate a TLS/SSL handshake, send to the server a TLS/SSL message obtained from the TLS/SSL layer encapsulated in TDS packet(s) of type PRELOGIN (0x12), and enter the "Sent TLS/SSL negotiation packet" state.
- If encryption was not negotiated and the upper layer did not request full encryption, the TDS client MUST send to the server a Login message with the authentication scheme specified by the user, and enter one of the following two states, depending on the message sent:
  - The "Sent LOGIN7 record with Complete Authentication Token" state if a login message with standard authentication or a login message with the FEDAUTH FeatureExt was sent.
  - "Sent LOGIN7 record with SPNEGO packet" state, if a login message with SPNEGO authentication was sent.

  The TDS specification does not prescribe the authentication protocol if SSPI [SSPI] authentication is used. The current implementation of SSPI supports NTLM [NTLM] and Kerberos [RFC4120].
- If encryption was not negotiated and the upper layer requested full encryption, then the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.
- If the response received from the server does not contain a structurally valid PRELOGIN response or it contains a structurally valid PRELOGIN response indicating an error, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.
- If NONCEOPT is specified in both the client PRELOGIN message and the server PRELOGIN message, the TDS client MUST maintain a state variable that includes the value of the NONCE that is sent to the server and a state variable that includes the value of the NONCE that is contained in the server's response.

Copyright © 2014 Microsoft Corporation.

3.2.5.2 Sent TLS/SSL Negotiation Packet State

If the response contains a structurally valid TLS/SSL response message (TDS packet **Type** 0x12), the TDS client MUST pass the TLS/SSL message contained in it to the TLS/SSL layer and MUST proceed as follows:

- If the TLS/SSL layer indicates that further handshake is needed, the TDS client MUST send to the server the TLS/SSL message obtained from the TLS/SSL layer encapsulated in TDS packet(s) of **Type** PRELOGIN (0x12).
- If the TLS/SSL layer indicates successful completion of the TLS/SSL handshake, the TDS client MUST send a Login message to the server with the authentication scheme specified by the user. The TDS client will then enter one of the following two states, depending on the message sent:
  - The "Sent LOGIN7 record with Complete Authentication Token" state, if a Login message with standard authentication or a Login message with the FEDAUTH FeatureId was sent.
  - The "Sent LOGIN7 record with SPNEGO packet" state, if a Login message with SPNEGO authentication was sent.

  The TDS specification does not prescribe the authentication protocol if SSPI [SSPI] authentication or federated authentication is used. The current implementation of SSPI supports NTLM [NTLM] and Kerberos [RFC4120].
- If login-only encryption was negotiated as described in section 2.2 in the PRELOGIN message description, then the first TDS packet of the Login message MUST be encrypted using TLS/SSL and encapsulated in a TLS/SSL message. All other TDS packets sent or received MUST be in plaintext.
- If full encryption was negotiated as described in section 2.2 in the PRELOGIN message description, then all subsequent TDS packets sent or received from this point on MUST be encrypted using TLS/SSL and encapsulated in a TLS/SSL message.
- If the TLS/SSL layer indicates an error, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

If the response received from the server does not contain a structurally valid TLS/SSL response or it contains a structurally valid response indicating an error, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.2.5.3 Sent LOGIN7 Record with Complete Authentication Token State

If the response received from the server contains a structurally valid Login response that indicates a successful login, and if the client used federated authentication to authenticate to the server, the client MUST read the Login response stream to find the FEATUREEXTACK token and find the FEDAUTH FeatureId. If the FEDAUTH FeatureId is not present, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the “Final State” state. If the FEDAUTH FeatureId is present, the client's action is based on the bFedAuthLibrary as follows:

- When the bFedAuthLibrary is Live ID Compact Token, the client MUST use the session key from its federated authentication token to compute the HMAC-SHA-256 [RFC6234] of the NONCE field in the FEDAUTH Feature Extension Acknowledgement, and the client MUST verify that the nonce matches the nonce sent by the client in its PRELOGIN request. If the signature field does not match the computed HMAC-SHA-256 or if the nonce does not match the nonce sent by the client

Copyright © 2014 Microsoft Corporation.

in its PRELOGIN request, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the “Final State” state.

- When the bFedAuthLibrary is Security Token and any of the following statements is true, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the “Final State” state:
  - The client had sent a nonce in the PRELOGIN message and either the NONCE field in FEDAUTH Feature Extension Acknowledgement is not present or the NONCE field does not match the nonce sent by the client in its PRELOGIN request.
  - The client had not sent a nonce in its PRELOGIN request, and there is a NONCE field present in the FEDAUTH Feature Extension Acknowledgement.

If the response received from the server contains a structurally valid Login response indicating a successful login and no Routing response is detected, the TDS client MUST indicate successful Login completion to the upper layer and enter the "Logged In" state.

If the response received from the server contains a structurally valid Login response indicating a successful login and also contains a routing response (a Routing ENVCHANGE token) after the LOGINACK token, the TDS client MUST enter the "Routing Completed" state.

If the response received from the server does not contain a structurally valid Login response or it contains a structurally valid Login response indicating login failure, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.2.5.4 Sent LOGIN7 Record with SPNEGO Packet State

If the response received from the server contains a structurally valid Login response indicating a successful login and no Routing response is detected, the TDS client MUST indicate successful Login completion to the upper layer and enter the "Logged In" state.

If the response received from the server contains a structurally valid Login response indicating a successful login and also contains a routing response (a Routing ENVCHANGE token) after the LOGINACK token, the TDS client MUST enter the "Routing Completed" state.

If the response received from the server contains a structurally valid SSPI response message, the TDS client MUST send to the server a SSPI message (TDS packet type %x11) containing the data obtained from the applicable SSPI layer. The TDS client SHOULD wait for the response and reenter this state when the response is received.

If the response received from the server does not contain a structurally valid Login response or SSPI response, or if it contains a structurally valid Login response indicating login failure, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.2.5.5 Logged In State

The TDS client waits for notification from the upper layer. If the upper layer requests a query to be sent to the server, the TDS client MUST send the appropriate request to the server and enter the "Sent Client Request" state. If MARS is enabled, the TDS client MUST send the appropriate request to the SMUX layer. If the upper layer requests a termination of the connection, the TDS client MUST disconnect from the server and enter the "Final State" state. If the TDS client detects a connection error from the transport layer, the TDS client MUST disconnect from the server and enter the "Final State" state.

Copyright © 2014 Microsoft Corporation.

3.2.5.6 Sent Client Request State

If the response received from the server contains a structurally valid response, the TDS client MUST indicate the result of the request to the upper layer and enter the "Logged In" state.

The client has the ability to return data/control to the upper layers while remaining in the "Sent Client Request" state while the complete response has not been received or processed.

If the TDS client supports Cancel Request and the upper layer requests a Cancel Request to be sent to the server, the TDS client will send an Attention message to the server, start the Cancel Timer, and enter the "Sent Attention" state.

If the response received from the server does not contain a structurally valid response, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.2.5.7 Sent Attention State

If the response is structurally valid and it does not acknowledge the Attention as described in section 2.2.1.6, then the TDS client MUST discard any data contained in the response and remain in the "Sent Attention" state.

If the response is structurally valid and it acknowledges the Attention as described in section 2.2.1.6, then the TDS client MUST discard any data contained in the response, indicate the completion of the query to the upper layer together with the cause of the Attention (either an upper-layer cancellation as described in section 3.2.4 or query timeout as described in section 3.2.2), and enter the "Logged In" state.

If the response received from the server is not structurally valid, then the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.2.5.8 Routing Completed State

The TDS client MUST:

- Read the rest of the login response from the server, processing the remaining tokens until the final DONE token is read, as it does with a normal login response.
- Discard all information read from the original login response except for the routing information supplied in the Routing ENVCHANGE token.
  - Any information in the original login response (for example, the language, collation, packet size, or database mirroring partner) will not apply to the subsequent connection established to the alternate server specified in the Routing ENVCHANGE token.
- Close the original connection, and enter the "Final State" state. The original connection cannot be used for any other purpose after the Routing ENVCHANGE token is read and the response is drained.

3.2.5.9 Final State

The connection is disconnected. All resources for this connection will be recycled by the TDS server.

Copyright © 2014 Microsoft Corporation.

3.2.6 Timer Events

If a TDS client implements the Connection Timer and the timer times out, then the TDS client MUST close the underlying connection, indicate the error to the upper layer, and enter the "Final State" state.

If a TDS client implements the Client Request Timer and the timer times out, then the TDS client MUST send an Attention message to the server and enter the "Sent Attention" state.

If a TDS client implements the Cancel Timer and the timer times out, then the TDS client MUST close the underlying connection, indicate the error to the upper layer, and enter the "Final State" state.

3.2.7 Other Local Events

Whenever an indication of a connection error is received from the underlying transport, the TDS client MUST close the transport connection, indicate an error to the upper layer, stop any timers if running, and enter the "Final State" state. If TCP is used as the underlying transport, examples of events that can trigger such action—dependent on the actual TCP implementation—might be media sense loss, a TCP connection going down in the middle of communication, or a TCP keep-alive failure.

3.3 Server Details

The following state machine diagram describes TDS on the server side.

Copyright © 2014 Microsoft Corporation.

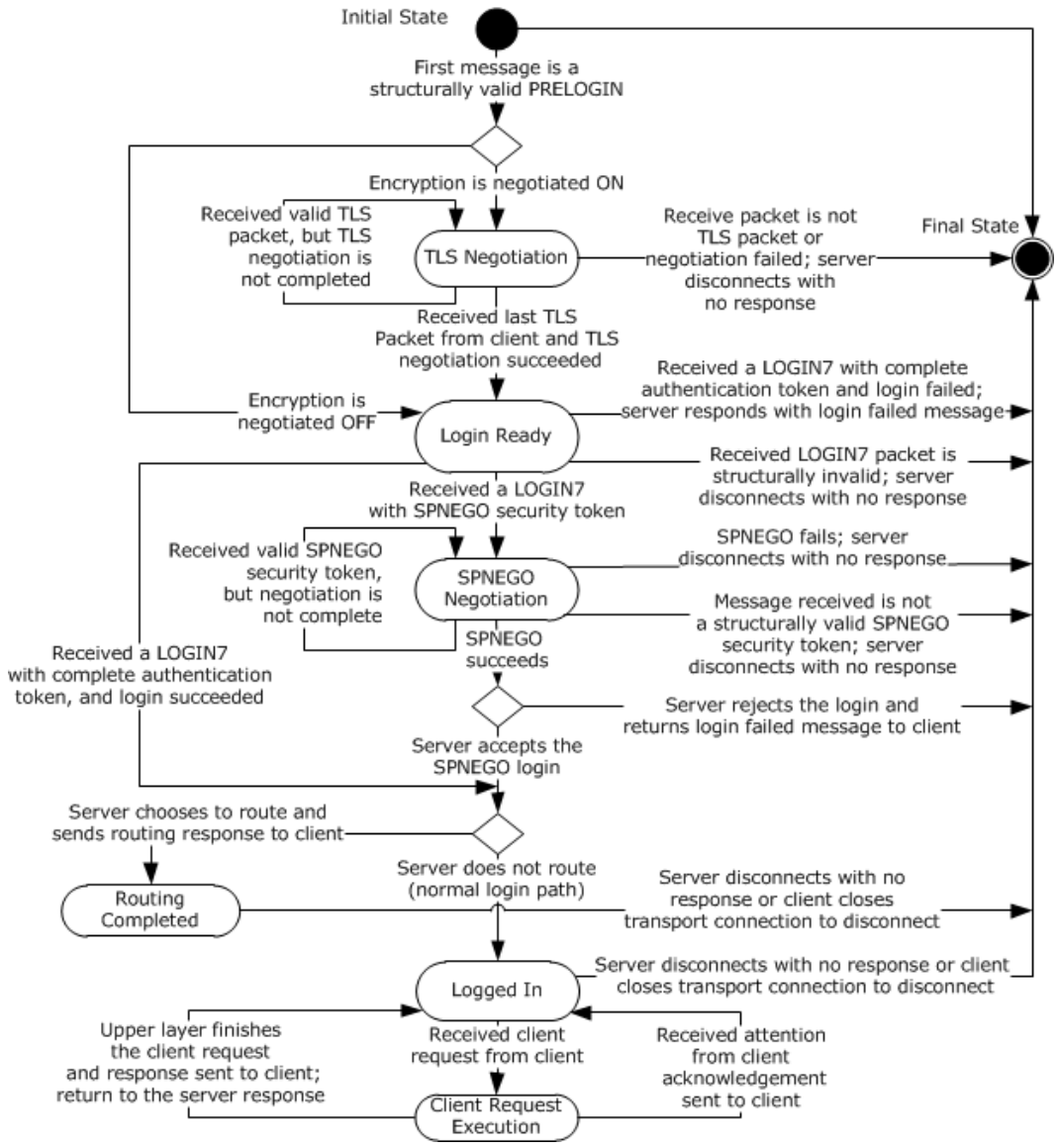

Figure 8: TDS server state machine

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this document.

The server SHOULD maintain the following states:

- Initial State
- TLS/SSL Negotiation State

Copyright © 2014 Microsoft Corporation.

- Login Ready State
- SPNEGO Negotiation State
- Logged In State
- Client Request Execution State
- Routing Completed State
- Final State

3.3.2 Timers

The TDS protocol does not regulate any timer on a data stream. The TDS server MAY implement a timer on any message found in section 2.

3.3.3 Initialization

The server MUST establish a listening endpoint based on one of the transport protocols described in section 2.1. The server can establish additional listening endpoints.

When a client makes a connection request, the transport layer listening endpoint will initialize all resources required for this connection. The server will be ready to receive a pre-login message.

3.3.4 Higher-Layer Triggered Events

A higher layer can choose to terminate a TDS connection at any time. In the current TDS implementation, the upper layer can kill a connection. When this happens, the server MUST terminate the connection and recycle all resources for this connection. No response will be sent to the client.

3.3.5 Message Processing Events and Sequencing Rules

The processing of messages received from a TDS client depends on the message type and the current state of the TDS server. The rest of this section describes the processing and actions to take on them. The message type is determined from the TDS packet type and the token stream inside the TDS packet payload, as described in section 2.2.

The corresponding action will be taken when the server in the following states.

3.3.5.1 Initial State

The TDS server receives the first packet from the client. The packet SHOULD be a PRELOGIN packet to set up context for login. A pre-login message is indicated by the PRELOGIN (0x12) message type described in section 2. The TDS server SHOULD close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state, if the first packet is not a structurally correct PRELOGIN packet or if the PRELOGIN packet does not contain the client version as the first option token. Otherwise, the TDS server MUST do one of the following:

- Return to the client a PRELOGIN structure wrapped in a table response (0x04) packet with Encryption and enter "TLS/SSL Negotiation" state if encryption is negotiated.
- Return to the client a PRELOGIN structure wrapped in a table response (0x04) packet without Encryption and enter unencrypted "Login Ready" state if encryption is not negotiated.

Copyright © 2014 Microsoft Corporation.

If a FEDAUTHREQUIRED option is contained in the PRELOGIN structure sent by the server to the client, the TDS server MUST maintain the value of the FEDAUTHREQUIRED option in a state variable to validate the LOGIN7 message with FEDAUTH FeatureId when the message arrives, as described in section 3.3.5.3.

If no FEDAUTHREQUIRED option is contained in the PRELOGIN structure sent by the server to the client, or if the value of B_FEDAUTHREQUIRED = 0, the TDS client can treat both events as equivalent and MUST remember the event in a state variable. Either state will be treated the same when the state variables are examined in the "Login Ready" state (see section 3.3.5.3 for further details).

If NONCEOPT is specified in both the client PRELOGIN message and the server PRELOGIN message, the TDS server MUST maintain a state variable that includes the values of both the NONCE it sent to the client and the NONCE the client sent to it during the PRELOGIN exchange.

3.3.5.2 TLS/SSL Negotiation State

If the next packet from the TDS client is not a TLS/SSL negotiation packet or the packet is not structurally correct, the TDS server SHOULD close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state. A TLS/SSL negotiation packet is a PRELOGIN (0x12) packet header encapsulated with TLS/SSL payload. The TDS server MUST exchange TLS/SSL negotiation packet with the client and reenter this state until the TLS/SSL negotiation is successfully completed. In this case, TDS server enters the "Login Ready" state.

3.3.5.3 Login Ready State

If the TDS server receives a valid LOGIN7 message with the FEDAUTH FeatureId from the client, the server MUST validate that one of the following is true:

- The TDS server's PRELOGIN structure contained a FEDAUTHREQUIRED option with the value 0x00, or the TDS server's PRELOGIN structure did not contain a FEDAUTHREQUIRED option, and the value of fFedAuthEcho is 0.
- The TDS server's PRELOGIN structure contained a FEDAUTHREQUIRED option with the value 0x01, and the value of fFedAuthEcho is 1.

If the TDS server receives a valid LOGIN7 message with the FEDAUTH FeatureId from the client but neither of the above statements is true, the server MUST send an ERROR packet, described in section 2, to the client. The TDS server MUST then close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state. Otherwise, the TDS server MUST process the FedAuthToken embedded in the packet in a way appropriate for the value of bFedAuthLibrary.

When the bFedAuthLibrary is Live ID Compact token, the TDS Server MUST respond as follows:

- If no NONCEOPT was specified in the client's PRELOGIN message, the TDS server MUST send a "Login failed" ERROR token to the client, the server MUST close the connection, and the server MUST enter the "Final State" state.
- If a NONCEOPT was specified in the client's PRELOGIN message, the federated authentication library layer responds with one of two results, and the TDS server continues processing according to the response as follows:
  - Success:

Copyright © 2014 Microsoft Corporation.

- The TDS server MUST use the session key from the federated authentication token to compute the HMAC-SHA-256 [RFC6234] of the data sent by the client. If the Signature field does not match the computed HMAC-SHA-256, or if the nonce does not match the nonce sent by the server in its PRELOGIN response, then the TDS server MUST send a "Login failed" ERROR token to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.

- If a ChannelBindingToken is present, the server MUST compare the ChannelBindingToken against the channel binding token calculated from the underlying TLS/SSL channel. If the two values do not match, then the TDS server MUST send a "Login failed" ERROR token to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.

- If both the channel binding token and the nonce match the expected values, the server MUST send the security token to the upper layer (typically an application that provides database management functions) for authorization. If the upper layer approves the security token, the TDS server MUST send a LOGINACK message that includes a FEATUREEXTACK token with the FEDAUTH FeatureId and immediately enter the "Logged In" state or enter the "Routing Completed" state if the server decides to route. If the upper layer rejects the security token, the TDS server MUST send a "Login failed" ERROR token to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.

- Error: The server then MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

When the bFedAuthLibrary is Security Token, the TDS server MUST respond as follows:

- If the server's PRELOGIN response contained a NONCEOPT, the TDS Server MUST validate to see whether the client's LOGIN7 packet has the same nonce echoed back as part of FEDAUTH Feature SignedData. If the NONCE field is not present or if the nonce does not match, the TDS server MUST send a "Login failed" ERROR token to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.

- If the server's PRELOGIN response did not contain a NONCEOPT, the TDS Server MUST verify that there is NO NONCE as part LOGIN7 FEDAUTH Feature SignedData. If a NONCE field is present, the TDS server MUST send a "Login failed" ERROR token back to the client, the TDS server MUST close the connection and the TDS server MUST enter the "Final State" state.

  - Success:

    - The server MUST send the security token to the upper layer (typically an application that provides database management functions) for authorization. If the upper layer approves the security token, the TDS server MUST send a LOGINACK message that includes a FEATUREEXTACK token with the FEDAUTH FeatureId and immediately enter the "Logged In" state or enter the "Routing Completed" state if the server decides to route. If the upper layer rejects the security token, the TDS server MUST send a "Login failed" ERROR token to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.

  - Error: The server then MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

If the TDS server receives a valid LOGIN7 packet with standard login, the TDS server MUST respond to the TDS client with a LOGINACK (0xAD) described in section 2 indicating login succeed. The TDS

Copyright © 2014 Microsoft Corporation.

server MUST enter the "Logged in" state or enter the "Routing Completed" state if the server decides to route.

If the TDS server receives a LOGIN7 packet with SSPI Negotiation packet, the TDS server MUST enter the "SPNEGO Negotiation" state.

If the TDS server receives a LOGIN7 packet with standard login packet, but the login is invalid, the TDS server MUST send an ERROR packet, described in section 2, to the client. The TDS server MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

If the packet received is not a structurally valid LOGIN7 packet, the TDS server will not send any response to the client. The TDS server MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.3.5.4 SPNEGO Negotiation State

This state is used to negotiate the security scheme between the client and server. The TDS server processes the packet received according to the following rules.

- If the packet received is a structurally valid SPNEGO [RFC4178] negotiation packet, the TDS server delegates processing of the security token embedded in the packet to the SPNEGO layer. The SPNEGO layer responds with one of three results, and the TDS server continues processing according to the response as follows:
  - Complete: The TDS server then sends the security token to the upper layer (typically an application that provides database management functions) for authorization. If the upper layer approves the security token, the TDS server returns the security token to the client within a LOGINACK message and immediately enters the "Logged In" state or enters the "Routing Completed" state if the server decides to route. If the upper layer rejects the security token, then a "Login failed" ERROR token is sent back to the client, the TDS server closes the connection, and the TDS server enters the "Final State" state.
  - Continue: The TDS server sends a SPNEGO [RFC4178] negotiation response to the client, embedding the new security token returned by SPNEGO as part of the Continue response. The server then waits for a message from the client and re-renters the SPNEGO negotiation state when such a packet is received.
  - Error: The server then MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.
- If the packet received is not a structurally valid SPNEGO [RFC4178] negotiation packet, the TDS server will send no response to the client. The TDS server MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

3.3.5.5 Logged In State

If a TDS of type 1, 3, 7, or 14 (see section 2.2.3.1.1) arrives, then the TDS server begins processing by raising an event to the upper layer containing the data of the client request and entering the Client Request Execution state. If any other TDS types arrive, then the server MUST enter the Final State state. The TDS server MUST continue to listen for messages from the client while awaiting notification of client request completion from the upper layer.

Copyright © 2014 Microsoft Corporation.

3.3.5.6 Client Request Execution State

The TDS server MUST continue to listen for messages from the client while awaiting notification of client request for completion from the upper layer. The TDS server MUST also do one of the following:

- If the upper layer notifies TDS that the client request has finished successfully, the TDS server MUST send the results in the formats described in section 2 to the TDS client and enter the "Logged In" state.
- If the upper layer notifies TDS that an error has been encountered during client request, the TDS server MUST send an error message (described in section 2) to the TDS client and enter the "Logged In" state.
- If an attention packet (described in section 2) is received during the execution of the current client request, it MUST deliver a cancel indication to the upper layer. If an attention packet (described in section 2) is received after the execution of the current client request, it SHOULD NOT deliver a cancel indication to the upper layer because there is no existing execution to cancel. The TDS server MUST send an attention acknowledgment to the TDS client and enter the "Logged In" state.
- If another client request packet is received during the execution of the current client request, the TDS server SHOULD queue the new client request, and continue processing the client request already in progress according to the preceding rules. When this operation is complete, the TDS server re-enters the "Client Request Execution" state and processes the newly arrived message.
- If MARS is enabled, all TDS server responses to client request messages MUST be passed through to the SMUX layer.
- If any other message type arrives, the server MUST close the connection and enter the "Final State" state.

3.3.5.7 Routing Completed State

The TDS server SHOULD wait for connection closure initiated by the client and enter the "Final State" state. If any request is received from the client in this state, the server SHOULD close the connection with no response and enter the "Final State" state.

3.3.5.8 Final State

The connection is disconnected. All resources for this connection will be recycled by the TDS server.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

When there is a failure in under-layers, the server SHOULD terminate the TDS session without sending any response to the client. The under-layer failure could be triggered by network failure. It can also be triggered by the termination action from the client, which could be communicated to the server stack by under-layers.

Copyright © 2014 Microsoft Corporation.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the TDS protocol. For each example, the binary TDS message is provided followed by the decomposition displayed in XML.

4.1 Pre-Login Request

Pre-Login request sent from the client to the server:

```
12 01 00 2F 00 00 01 00 00 00 1A 00 06 01 00 20
00 01 02 00 21 00 01 03 00 22 00 04 04 00 26 00
01 FF 09 00 00 00 00 00 01 00 B8 0D 00 00 01


<PacketHeader>
    <Type>
      <BYTE>12 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>2F </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <Prelogin>
      <TokenType>
         <BYTE>00 </BYTE>
      </TokenType>
      <TokenPosition>
         <USHORT>00 1A</USHORT>
      </TokenPosition>
      <TokenLeng>
         <USHORT>00 06</USHORT>
      </TokenLeng>
      <TokenType>
         <BYTE>01 </BYTE>
      </TokenType>
      <TokenPosition>
         <USHORT>00 20</USHORT>
      </TokenPosition>
      <TokenLeng>
         <USHORT>00 01</USHORT>
      </TokenLeng>
      <TokenType>
```


Copyright © 2014 Microsoft Corporation.

```
        <BYTE>02 </BYTE>
      </TokenType>
      <TokenPosition>
        <USHORT>00 21</USHORT>
      </TokenPosition>
      <TokenLeng>
        <USHORT>00 01</USHORT>
      </TokenLeng>
      <TokenType>
        <BYTE>03 </BYTE>
      </TokenType>
      <TokenPosition>
        <USHORT>00 22</USHORT>
      </TokenPosition>
      <TokenLeng>
        <USHORT>00 04</USHORT>
      </TokenLeng>
      <TokenType>
        <BYTE>04 </BYTE>
      </TokenType>
      <TokenPosition>
        <USHORT>00 26</USHORT>
      </TokenPosition>
      <TokenLeng>
        <USHORT>00 01</USHORT>
      </TokenLeng>
      <TokenType>
        <BYTE>FF </BYTE>
      </TokenType>
      <PreloginData>
        <BYTES>09 00 00 00 00 00 01 00 B8 0D 00 00 01</BYTE>
      </PreloginData>
    </Prelogin>
  </PacketData>
```

4.2 Login Request

LOGIN7 stream sent from the client to the server:

```
10 01 00 90 00 00 01 00 88 00 00 00 02 00 09 72
00 10 00 00 00 00 00 07 00 01 00 00 00 00 00 00
E0 03 00 00 E0 01 00 00 09 04 00 00 5E 00 08 00
6E 00 02 00 72 00 00 00 72 00 07 00 80 00 00 00
80 00 00 00 80 00 04 00 88 00 00 00 88 00 00 00
00 50 8B E2 B7 8F 88 00 00 00 88 00 00 00 88 00
00 00 00 00 00 00 73 00 6B 00 6F 00 73 00 74 00
6F 00 76 00 31 00 73 00 61 00 4F 00 53 00 51 00
4C 00 2D 00 33 00 32 00 4F 00 44 00 42 00 43 00
```

```
  <PacketHeader>
    <Type>
      <BYTE>10 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
```


Copyright © 2014 Microsoft Corporation.

```
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>90 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <Login7>
      <Length>
        <DWORD>88 00 00 00 </DWORD>
      </Length>
      <TDSVersion>
        <DWORD>02 00 09 72 </DWORD>
      </TDSVersion>
      <PacketSize>
        <DWORD>00 10 00 00 </DWORD>
      </PacketSize>
      <ClientProgVer>
        <DWORD>00 00 00 07 </DWORD>
      </ClientProgVer>
      <ClientPID>
        <DWORD>00 01 00 00 </DWORD>
      </ClientPID>
      <ConnectionID>
        <DWORD>00 00 00 00 </DWORD>
      </ConnectionID>
      <OptionFlags1>
        <BYTE>E0 </BYTE>
      </OptionFlags1>
      <OptionFlags2>
        <BYTE>03 </BYTE>
      </OptionFlags2>
      <TypeFlags>
        <BYTE>00 </BYTE>
      </TypeFlags>
      <OptionFlags3>
        <BYTE>00 </BYTE>
      </OptionFlags3>
      <ClientTimZone>
        <LONG>E0 01 00 00 </LONG>
      </ClientTimZone>
      <ClientLCID>
        <DWORD>09 04 00 00 </DWORD>
      </ClientLCID>
      <OffsetLength>
        <ibHostName>
          <USHORT>5E 00 </USHORT>
        </ibHostName>
        <cchHostName>
          <USHORT>08 00 </USHORT>
```


Copyright © 2014 Microsoft Corporation.

```
</cchHostName>
<ibUserName>
  <USHORT>6E 00 </USHORT>
</ibUserName>
<cchUserName>
  <USHORT>02 00 </USHORT>
</cchUserName>
<ibPassword>
  <USHORT>72 00 </USHORT>
</ibPassword>
<cchPassword>
  <USHORT>00 00 </USHORT>
</cchPassword>
<ibAppName>
  <USHORT>72 00 </USHORT>
</ibAppName>
<cchAppName>
  <USHORT>07 00 </USHORT>
</cchAppName>
<ibServerName>
  <USHORT>80 00 </USHORT>
</ibServerName>
<cchServerName>
  <USHORT>00 00 </USHORT>
</cchServerName>
<ibUnused>
  <USHORT>80 00 </USHORT>
</ibUnused>
<cbUnused>
  <USHORT>00 00 </USHORT>
</cbUnused>
<ibCltIntName>
  <USHORT>80 00 </USHORT>
</ibCltIntName>
<cchCltIntName>
  <USHORT>04 00 </USHORT>
</cchCltIntName>
<ibLanguage>
  <USHORT>88 00 </USHORT>
</ibLanguage>
<cchLanguage>
  <USHORT>00 00 </USHORT>
</cchLanguage>
<ibDatabase>
  <USHORT>88 00 </USHORT>
</ibDatabase>
<cchDatabase>
  <USHORT>00 00 </USHORT>
</cchDatabase>
<ClientID>
  <BYTES>00 50 8B E2 B7 8F </BYTES>
</ClientID>
<ibSSPI>
  <USHORT>88 00 </USHORT>
</ibSSPI>
<cbSSPI>
  <USHORT>00 00 </USHORT>
</cbSSPI>
<ibAtchDBFile>
```


Copyright © 2014 Microsoft Corporation.

```
          <USHORT>88 00 </USHORT>
        </ibAtchDBFile>
        <cchAtchDBFile>
          <USHORT>00 00 </USHORT>
        </cchAtchDBFile>
        <ibChangePassword>
          <USHORT>88 00 </USHORT>
        </ibChangePassword>
        <cchChangePassword>
          <USHORT>00 00 </USHORT>
        </cchChangePassword>
        <cbSSPILong>
          <LONG>00 00 00 00 </LONG>
        </cbSSPILong>
      </OffsetLength>
      <Data>
        <BYTES>73 00 6B 00 6F 00 73 00 74 00 6F 00 76 00 31 00 73 00 61 00
4F 00 53 00 51 00 4C 00 2D 00 33 00 32 00 4F 00 44 00 42 00 43 00 </BYTES>
      </Data>
    </Login7>
  </PacketData>
```

4.3 Login Request with Federated Authentication

LOGIN7 stream sent from the client to the server, including the Feature Extensions block with the Federated Authentication feature:

```
10 01 08 10 00 00 01 00 08 08 00 00 04 00 00 74
00 10 00 00 00 00 00 07 40 37 00 00 00 00 00 00
E0 03 00 10 E0 01 00 00 09 04 00 00 5E 00 0F 00
00 00 00 00 00 00 00 00 7C 00 06 00 88 00 18 00
B8 00 04 00 BC 00 04 00 C4 00 00 00 C4 00 00 00
00 15 5D 71 E7 42 00 00 00 00 C4 00 00 00 00 00
00 00 00 00 00 00 44 00 41 00 4E 00 42 00 45 00
4E 00 45 00 44 00 33 00 2D 00 58 00 47 00 5A 00
55 00 4F 00 53 00 51 00 4C 00 43 00 4D 00 44 00
63 00 6C 00 6F 00 75 00 64 00 2E 00 64 00 65 00
76 00 2E 00 6D 00 73 00 63 00 64 00 73 00 2E 00
63 00 6F 00 6D 00 2C 00 31 00 34 00 33 00 35 00
C4 00 00 00 4F 00 44 00 42 00 43 00 02 3E 07 00
00 01 E2 06 00 00 74 00 3D 00 45 00 77 00 43 00
51 00 41 00 6A 00 4B 00 6A 00 42 00 77 00 41 00
55 00 53 00 30 00 48 00 6F 00 2F 00 30 00 65 00
31 00 52 00 35 00 4D 00 32 00 77 00 37 00 74 00
69 00 44 00 72 00 36 00 70 00 4D 00 63 00 75 00
77 00 33 00 35 00 6B 00 41 00 41 00 61 00 62 00
76 00 45 00 38 00 6F 00 55 00 45 00 71 00 2F 00
50 00 45 00 75 00 4C 00 37 00 32 00 64 00 54 00
74 00 64 00 4D 00 42 00 43 00 7A 00 46 00 37 00
41 00 64 00 4A 00 6D 00 6E 00 70 00 45 00 68 00
34 00 32 00 65 00 41 00 46 00 57 00 37 00 2F 00
38 00 49 00 6A 00 41 00 53 00 6D 00 78 00 79 00
50 00 30 00 30 00 71 00 59 00 64 00 76 00 65 00
31 00 5A 00 6B 00 45 00 2F 00 58 00 6C 00 63 00
4E 00 79 00 69 00 77 00 6F 00 55 00 76 00 54 00
4E 00 4F 00 6B 00 74 00 42 00 45 00 4C 00 71 00
76 00 37 00 58 00 4A 00 54 00 6F 00 47 00 71 00
```


Copyright © 2014 Microsoft Corporation.

```
56 00 32 00 6F 00 6B 00 51 00 6E 00 2F 00 65 00
63 00 50 00 41 00 78 00 32 00 71 00 6A 00 55 00
57 00 74 00 6B 00 54 00 59 00 56 00 66 00 62 00
72 00 51 00 58 00 78 00 76 00 4E 00 58 00 66 00
69 00 72 00 77 00 47 00 57 00 63 00 43 00 45 00
4B 00 79 00 46 00 64 00 76 00 35 00 62 00 78 00
55 00 75 00 68 00 69 00 49 00 62 00 39 00 42 00
55 00 56 00 54 00 56 00 4B 00 6A 00 57 00 51 00
34 00 78 00 4E 00 46 00 4E 00 6B 00 43 00 33 00
73 00 58 00 71 00 6F 00 46 00 52 00 31 00 49 00
46 00 64 00 76 00 6A 00 4C 00 76 00 63 00 71 00
71 00 38 00 70 00 47 00 69 00 54 00 64 00 51 00
2F 00 76 00 37 00 6F 00 44 00 6B 00 38 00 78 00
5A 00 42 00 6E 00 61 00 73 00 4C 00 36 00 71 00
2F 00 62 00 38 00 36 00 73 00 5A 00 6C 00 2B 00
55 00 35 00 77 00 69 00 4C 00 50 00 50 00 54 00
68 00 67 00 4D 00 70 00 49 00 46 00 2F 00 42 00
61 00 43 00 2F 00 72 00 45 00 4F 00 51 00 50 00
6B 00 6B 00 63 00 51 00 4A 00 4F 00 56 00 54 00
71 00 51 00 72 00 64 00 63 00 4B 00 4F 00 4D 00
56 00 4E 00 4A 00 61 00 55 00 50 00 49 00 52 00
33 00 64 00 2B 00 42 00 62 00 43 00 66 00 31 00
6D 00 5A 00 62 00 7A 00 6F 00 6E 00 64 00 4F 00
51 00 77 00 39 00 57 00 37 00 49 00 77 00 63 00
6D 00 30 00 45 00 44 00 69 00 78 00 51 00 71 00
70 00 41 00 74 00 61 00 75 00 63 00 42 00 48 00
4F 00 33 00 46 00 75 00 37 00 6D 00 49 00 63 00
69 00 78 00 46 00 39 00 67 00 53 00 2F 00 46 00
4B 00 38 00 2F 00 37 00 6C 00 6E 00 78 00 34 00
53 00 6E 00 33 00 5A 00 2F 00 51 00 7A 00 2B 00
48 00 76 00 6A 00 4C 00 49 00 57 00 2F 00 76 00
44 00 77 00 7A 00 77 00 44 00 5A 00 67 00 41 00
41 00 43 00 4E 00 6D 00 73 00 64 00 34 00 65 00
79 00 68 00 4E 00 49 00 75 00 59 00 41 00 47 00
62 00 58 00 62 00 41 00 4B 00 51 00 37 00 63 00
4A 00 66 00 31 00 6B 00 78 00 31 00 31 00 6D 00
4E 00 42 00 49 00 34 00 79 00 42 00 44 00 36 00
4E 00 4E 00 54 00 53 00 6F 00 63 00 75 00 46 00
42 00 6A 00 4F 00 6B 00 2B 00 73 00 41 00 5A 00
4C 00 35 00 5A 00 34 00 56 00 4E 00 32 00 4D 00
4C 00 71 00 49 00 35 00 71 00 38 00 58 00 54 00
58 00 35 00 72 00 58 00 57 00 65 00 79 00 4B 00
75 00 62 00 76 00 49 00 2F 00 59 00 2F 00 6D 00
64 00 42 00 6A 00 64 00 4E 00 36 00 37 00 51 00
57 00 49 00 72 00 75 00 65 00 73 00 4B 00 52 00
56 00 65 00 32 00 35 00 31 00 31 00 54 00 6E 00
42 00 58 00 46 00 62 00 70 00 53 00 47 00 6E 00
42 00 4A 00 65 00 38 00 6F 00 31 00 4B 00 69 00
50 00 55 00 70 00 55 00 4B 00 6A 00 48 00 74 00
54 00 6D 00 75 00 34 00 36 00 4F 00 43 00 4F 00
38 00 4A 00 49 00 62 00 48 00 47 00 6C 00 70 00
4F 00 70 00 62 00 70 00 6E 00 50 00 49 00 73 00
79 00 36 00 39 00 49 00 58 00 6B 00 45 00 7A 00
67 00 38 00 64 00 36 00 34 00 43 00 74 00 65 00
54 00 2F 00 61 00 63 00 73 00 35 00 68 00 4F 00
72 00 62 00 32 00 44 00 73 00 44 00 6F 00 6E 00
4A 00 71 00 65 00 6D 00 46 00 58 00 54 00 47 00
6D 00 35 00 30 00 37 00 30 00 65 00 71 00 35 00
74 00 75 00 6A 00 69 00 5A 00 75 00 43 00 52 00
```


Copyright © 2014 Microsoft Corporation.

```
30 00 4C 00 54 00 36 00 62 00 59 00 49 00 76 00
59 00 64 00 76 00 42 00 6D 00 70 00 52 00 6C 00
31 00 33 00 38 00 41 00 53 00 42 00 32 00 58 00
48 00 30 00 36 00 43 00 4B 00 39 00 33 00 55 00
38 00 63 00 4F 00 54 00 38 00 68 00 68 00 6E 00
6A 00 32 00 53 00 4D 00 49 00 61 00 2B 00 4B 00
30 00 37 00 62 00 72 00 6F 00 66 00 43 00 50 00
4B 00 35 00 37 00 51 00 66 00 43 00 4D 00 35 00
32 00 4F 00 49 00 4B 00 33 00 2F 00 30 00 67 00
70 00 6F 00 4C 00 41 00 6C 00 4A 00 6E 00 49 00
56 00 59 00 72 00 45 00 6B 00 6B 00 41 00 6E 00
47 00 59 00 79 00 70 00 52 00 34 00 45 00 34 00
6F 00 4D 00 4D 00 33 00 63 00 48 00 32 00 38 00
4D 00 42 00 4C 00 66 00 30 00 76 00 47 00 57 00
2F 00 4D 00 62 00 6E 00 74 00 61 00 31 00 35 00
47 00 37 00 47 00 72 00 71 00 76 00 54 00 41 00
73 00 5A 00 30 00 78 00 42 00 6C 00 35 00 76 00
38 00 74 00 44 00 31 00 4F 00 70 00 70 00 6B 00
47 00 6D 00 2B 00 78 00 56 00 62 00 54 00 78 00
49 00 6E 00 2F 00 31 00 7A 00 51 00 67 00 67 00
32 00 71 00 54 00 4F 00 45 00 53 00 6D 00 6E 00
63 00 49 00 4B 00 38 00 49 00 44 00 6A 00 46 00
50 00 7A 00 79 00 59 00 31 00 71 00 38 00 45 00
49 00 6A 00 38 00 39 00 52 00 72 00 2B 00 63 00
6C 00 51 00 4C 00 33 00 62 00 76 00 39 00 69 00
38 00 4F 00 34 00 68 00 77 00 66 00 43 00 62 00
5A 00 51 00 65 00 63 00 38 00 59 00 4F 00 31 00
4D 00 67 00 4F 00 5A 00 32 00 63 00 2B 00 5A 00
74 00 39 00 45 00 55 00 54 00 67 00 31 00 78 00
51 00 49 00 31 00 6A 00 49 00 71 00 76 00 2B 00
65 00 47 00 63 00 62 00 47 00 50 00 66 00 47 00
78 00 4C 00 4C 00 45 00 57 00 76 00 66 00 2B 00
6A 00 54 00 71 00 58 00 63 00 6B 00 55 00 68 00
79 00 4A 00 42 00 47 00 33 00 6D 00 41 00 45 00
3D 00 26 00 70 00 3D 00 DF 31 12 79 58 7C 0F CD
2B ED 31 0D 8A 06 71 A7 C4 A6 DE BB 08 4F 37 12
07 E2 E9 09 C7 B0 2A 1D 74 6C 73 2D 75 6E 69 71
75 65 3A 20 87 E9 38 E3 2C 32 4F 6C F9 8E 5B 0D
40 B2 1B 57 0E AB FE EF 2F CD 58 DA 0F 7F BD 9C
D0 8B F2 15 85 F3 83 20 7C 22 F8 17 9A 95 8F FF
```

```
<PacketHeader>
  <Type>
    <BYTE>10 </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>08 </BYTE>
    <BYTE>10 </BYTE>
  </Length>
  <SPID>
    <BYTE>00 </BYTE>
    <BYTE>00 </BYTE>
  </SPID>
  <Packet>
    <BYTE>01 </BYTE>
  </Packet>
```


Copyright © 2014 Microsoft Corporation.

```
      <Window>
        <BYTE>00 </BYTE>
      </Window>
    </PacketHeader>
    <PacketData>
      <Login7>
        <Length>
          <DWORD>08 08 00 00 </DWORD>
        </Length>
        <TDSVersion>
          <DWORD>04 00 00 74 </DWORD>
        </TDSVersion>
        <PacketSize>
          <DWORD>00 10 00 00 </DWORD>
        </PacketSize>
        <ClientProgVer>
          <DWORD>00 00 00 07 </DWORD>
        </ClientProgVer>
        <ClientPID>
          <DWORD>40 37 00 00 </DWORD>
        </ClientPID>
        <ConnectionID>
          <DWORD>00 00 00 00 </DWORD>
        </ConnectionID>
        <OptionFlags1>
          <BYTE>E0 </BYTE>
        </OptionFlags1>
        <OptionFlags2>
          <BYTE>03 </BYTE>
        </OptionFlags2>
        <TypeFlags>
          <BYTE>00 </BYTE>
        </TypeFlags>
        <OptionFlags3>
          <BYTE>10 </BYTE>
        </OptionFlags3>
        <ClientTimZone>
          <LONG>E0 01 00 00 </LONG>
        </ClientTimZone>
        <ClientLCID>
          <DWORD>09 04 00 00 </DWORD>
        </ClientLCID>
        <OffsetLength>
          <ibHostName>
            <USHORT>5E 00 </USHORT>
          </ibHostName>
          <cchHostName>
            <USHORT>0F 00 </USHORT>
          </cchHostName>
          <ibUserName>
            <USHORT>00 00 </USHORT>
          </ibUserName>
          <cchUserName>
            <USHORT>00 00 </USHORT>
          </cchUserName>
          <ibPassword>
            <USHORT>00 00 </USHORT>
          </ibPassword>
          <cchPassword>
```


Copyright © 2014 Microsoft Corporation.

```
  <USHORT>00 00 </USHORT>
</cchPassword>
<ibAppName>
  <USHORT>7C 00 </USHORT>
</ibAppName>
<cchAppName>
  <USHORT>06 00 </USHORT>
</cchAppName>
<ibServerName>
  <USHORT>88 00 </USHORT>
</ibServerName>
<cchServerName>
  <USHORT>18 00 </USHORT>
</cchServerName>
<ibExtension>
  <USHORT>B8 00 </USHORT>
</ibExtension>
<cbExtension>
  <USHORT>04 00 </USHORT>
</cbExtension>
<ibCltIntName>
  <USHORT>BC 00 </USHORT>
</ibCltIntName>
<cchCltIntName>
  <USHORT>04 00 </USHORT>
</cchCltIntName>
<ibLanguage>
  <USHORT>C4 00 </USHORT>
</ibLanguage>
<cchLanguage>
  <USHORT>00 00 </USHORT>
</cchLanguage>
<ibDatabase>
  <USHORT>C4 00 </USHORT>
</ibDatabase>
<cchDatabase>
  <USHORT>00 00 </USHORT>
</cchDatabase>
<ClientID>
  <BYTES>00 15 5D 71 E7 42 </BYTES>
</ClientID>
<ibSSPI>
  <USHORT>00 00 </USHORT>
</ibSSPI>
<cbSSPI>
  <USHORT>00 00 </USHORT>
</cbSSPI>
<ibAtchDBFile>
  <USHORT>C4 00 </USHORT>
</ibAtchDBFile>
<cchAtchDBFile>
  <USHORT>00 00 </USHORT>
</cchAtchDBFile>
<ibChangePassword>
  <USHORT>00 00 </USHORT>
</ibChangePassword>
<cchChangePassword>
  <USHORT>00 00 </USHORT>
</cchChangePassword>
```


Copyright © 2014 Microsoft Corporation.

```
          <cbSSPILong>
            <LONG>00 00 00 00 </LONG>
          </cbSSPILong>
        </OffsetLength>
        <Data>
          <BYTES>44 00 41 00 4E 00 42 00 45 00
  4E 00 45 00 44 00 33 00 2D 00 58 00 47 00 5A 00
  55 00 4F 00 53 00 51 00 4C 00 43 00 4D 00 44 00
  63 00 6C 00 6F 00 75 00 64 00 2E 00 64 00 65 00
  76 00 2E 00 6D 00 73 00 63 00 64 00 73 00 2E 00
  63 00 6F 00 6D 00 2C 00 31 00 34 00 33 00 35 00
  C4 00 00 00 4F 00 44 00 42 00 43 00 </BYTES>
        </Data>
        <FeatureExt>
          <FeatureOpt>
            <FeatureId>
              <BYTE>02 </BYTE>
            </FeatureId>
            <FeatureDataLen>
              <DWORD>3E 07 00 00 </DWORD>
            </FeatureDataLen>
            <FeatureData>
              <Options>
                <BYTE>01 </BYTE>
              </Options>
              <FedAuthToken>
                <L_VARBYTE>
                  <LONGLEN>
                    <DWORD>E2 06 00 00 </DWORD>
                  </LONGLEN>
                  <BYTES>74 00 3D 00 45 00 77 00 43 00
  51 00 41 00 6A 00 4B 00 6A 00 42 00 77 00 41 00
  55 00 53 00 30 00 48 00 6F 00 2F 00 30 00 65 00
  31 00 52 00 35 00 4D 00 32 00 77 00 37 00 74 00
  69 00 44 00 72 00 36 00 70 00 4D 00 63 00 75 00
  77 00 33 00 35 00 6B 00 41 00 41 00 61 00 62 00
  76 00 45 00 38 00 6F 00 55 00 45 00 71 00 2F 00
  50 00 45 00 75 00 4C 00 37 00 32 00 64 00 54 00
  74 00 64 00 4D 00 42 00 43 00 7A 00 46 00 37 00
  41 00 64 00 4A 00 6D 00 6E 00 70 00 45 00 68 00
  34 00 32 00 65 00 41 00 46 00 57 00 37 00 2F 00
  38 00 49 00 6A 00 41 00 53 00 6D 00 78 00 79 00
  50 00 30 00 30 00 71 00 59 00 64 00 76 00 65 00
  31 00 5A 00 6B 00 45 00 2F 00 58 00 6C 00 63 00
  4E 00 79 00 69 00 77 00 6F 00 55 00 76 00 54 00
  4E 00 4F 00 6B 00 74 00 42 00 45 00 4C 00 71 00
  76 00 37 00 58 00 4A 00 54 00 6F 00 47 00 71 00
  56 00 32 00 6F 00 6B 00 51 00 6E 00 2F 00 65 00
  63 00 50 00 41 00 78 00 32 00 71 00 6A 00 55 00
  57 00 74 00 6B 00 54 00 59 00 56 00 66 00 62 00
  72 00 51 00 58 00 78 00 76 00 4E 00 58 00 66 00
  69 00 72 00 77 00 47 00 57 00 63 00 43 00 45 00
  4B 00 79 00 46 00 64 00 76 00 35 00 62 00 78 00
  55 00 75 00 68 00 69 00 49 00 62 00 39 00 42 00
  55 00 56 00 54 00 56 00 4B 00 6A 00 57 00 51 00
  34 00 78 00 4E 00 46 00 4E 00 6B 00 43 00 33 00
  73 00 58 00 71 00 6F 00 46 00 52 00 31 00 49 00
  46 00 64 00 76 00 6A 00 4C 00 76 00 63 00 71 00
  71 00 38 00 70 00 47 00 69 00 54 00 64 00 51 00
```


Copyright © 2014 Microsoft Corporation.

```
2F 00 76 00 37 00 6F 00 44 00 6B 00 38 00 78 00
5A 00 42 00 6E 00 61 00 73 00 4C 00 36 00 71 00
2F 00 62 00 38 00 36 00 73 00 5A 00 6C 00 2B 00
55 00 35 00 77 00 69 00 4C 00 50 00 50 00 54 00
68 00 67 00 4D 00 70 00 49 00 46 00 2F 00 42 00
61 00 43 00 2F 00 72 00 45 00 4F 00 51 00 50 00
6B 00 6B 00 63 00 51 00 4A 00 4F 00 56 00 54 00
71 00 51 00 72 00 64 00 63 00 4B 00 4F 00 4D 00
56 00 4E 00 4A 00 61 00 55 00 50 00 49 00 52 00
33 00 64 00 2B 00 42 00 62 00 43 00 66 00 31 00
6D 00 5A 00 62 00 7A 00 6F 00 6E 00 64 00 4F 00
51 00 77 00 39 00 57 00 37 00 49 00 77 00 63 00
6D 00 30 00 45 00 44 00 69 00 78 00 51 00 71 00
70 00 41 00 74 00 61 00 75 00 63 00 42 00 48 00
4F 00 33 00 46 00 75 00 37 00 6D 00 49 00 63 00
69 00 78 00 46 00 39 00 67 00 53 00 2F 00 46 00
4B 00 38 00 2F 00 37 00 6C 00 6E 00 78 00 34 00
53 00 6E 00 33 00 5A 00 2F 00 51 00 7A 00 2B 00
48 00 76 00 6A 00 4C 00 49 00 57 00 2F 00 76 00
44 00 77 00 7A 00 77 00 44 00 5A 00 67 00 41 00
41 00 43 00 4E 00 6D 00 73 00 64 00 34 00 65 00
79 00 68 00 4E 00 49 00 75 00 59 00 41 00 47 00
62 00 58 00 62 00 41 00 4B 00 51 00 37 00 63 00
4A 00 66 00 31 00 6B 00 78 00 31 00 31 00 6D 00
4E 00 42 00 49 00 34 00 79 00 42 00 44 00 36 00
4E 00 4E 00 54 00 53 00 6F 00 63 00 75 00 46 00
42 00 6A 00 4F 00 6B 00 2B 00 73 00 41 00 5A 00
4C 00 35 00 5A 00 34 00 56 00 4E 00 32 00 4D 00
4C 00 71 00 49 00 35 00 71 00 38 00 58 00 54 00
58 00 35 00 72 00 58 00 57 00 65 00 79 00 4B 00
75 00 62 00 76 00 49 00 2F 00 59 00 2F 00 6D 00
64 00 42 00 6A 00 64 00 4E 00 36 00 37 00 51 00
57 00 49 00 72 00 75 00 65 00 73 00 4B 00 52 00
56 00 65 00 32 00 35 00 31 00 31 00 54 00 6E 00
42 00 58 00 46 00 62 00 70 00 53 00 47 00 6E 00
42 00 4A 00 65 00 38 00 6F 00 31 00 4B 00 69 00
50 00 55 00 70 00 55 00 4B 00 6A 00 48 00 74 00
54 00 6D 00 75 00 34 00 36 00 4F 00 43 00 4F 00
38 00 4A 00 49 00 62 00 48 00 47 00 6C 00 70 00
4F 00 70 00 62 00 70 00 6E 00 50 00 49 00 73 00
79 00 36 00 39 00 49 00 58 00 6B 00 45 00 7A 00
67 00 38 00 64 00 36 00 34 00 43 00 74 00 65 00
54 00 2F 00 61 00 63 00 73 00 35 00 68 00 4F 00
72 00 62 00 32 00 44 00 73 00 44 00 6F 00 6E 00
4A 00 71 00 65 00 6D 00 46 00 58 00 54 00 47 00
6D 00 35 00 30 00 37 00 30 00 65 00 71 00 35 00
74 00 75 00 6A 00 69 00 5A 00 75 00 43 00 52 00
30 00 4C 00 54 00 36 00 62 00 59 00 49 00 76 00
59 00 64 00 76 00 42 00 6D 00 70 00 52 00 6C 00
31 00 33 00 38 00 41 00 53 00 42 00 32 00 58 00
48 00 30 00 36 00 43 00 4B 00 39 00 33 00 55 00
38 00 63 00 4F 00 54 00 38 00 68 00 68 00 6E 00
6A 00 32 00 53 00 4D 00 49 00 61 00 2B 00 4B 00
30 00 37 00 62 00 72 00 6F 00 66 00 43 00 50 00
4B 00 35 00 37 00 51 00 66 00 43 00 4D 00 35 00
32 00 4F 00 49 00 4B 00 33 00 2F 00 30 00 67 00
70 00 6F 00 4C 00 41 00 6C 00 4A 00 6E 00 49 00
56 00 59 00 72 00 45 00 6B 00 6B 00 41 00 6E 00
47 00 59 00 79 00 70 00 52 00 34 00 45 00 34 00
```


Copyright © 2014 Microsoft Corporation.

```
6F 00 4D 00 4D 00 33 00 63 00 48 00 32 00 38 00
4D 00 42 00 4C 00 66 00 30 00 76 00 47 00 57 00
2F 00 4D 00 62 00 6E 00 74 00 61 00 31 00 35 00
47 00 37 00 47 00 72 00 71 00 76 00 54 00 41 00
73 00 5A 00 30 00 78 00 42 00 6C 00 35 00 76 00
38 00 74 00 44 00 31 00 4F 00 70 00 70 00 6B 00
47 00 6D 00 2B 00 78 00 56 00 62 00 54 00 78 00
49 00 6E 00 2F 00 31 00 7A 00 51 00 67 00 67 00
32 00 71 00 54 00 4F 00 45 00 53 00 6D 00 6E 00
63 00 49 00 4B 00 38 00 49 00 44 00 6A 00 46 00
50 00 7A 00 79 00 59 00 31 00 71 00 38 00 45 00
49 00 6A 00 38 00 39 00 52 00 72 00 2B 00 63 00
6C 00 51 00 4C 00 33 00 62 00 76 00 39 00 69 00
38 00 4F 00 34 00 68 00 77 00 66 00 43 00 62 00
5A 00 51 00 65 00 63 00 38 00 59 00 4F 00 31 00
4D 00 67 00 4F 00 5A 00 32 00 63 00 2B 00 5A 00
74 00 39 00 45 00 55 00 54 00 67 00 31 00 78 00
51 00 49 00 31 00 6A 00 49 00 71 00 76 00 2B 00
65 00 47 00 63 00 62 00 47 00 50 00 66 00 47 00
78 00 4C 00 4C 00 45 00 57 00 76 00 66 00 2B 00
6A 00 54 00 71 00 58 00 63 00 6B 00 55 00 68 00
79 00 4A 00 42 00 47 00 33 00 6D 00 41 00 45 00
3D 00 26 00 70 00 3D 00 </BYTES>
              </L_VARBYTE>
            </FedAuthToken>
            <SignedData>
              <Nonce>
                <BYTES>DF 31 12 79 58 7C 0F CD
2B ED 31 0D 8A 06 71 A7 C4 A6 DE BB 08 4F 37 12
07 E2 E9 09 C7 B0 2A 1D </BYTES>
              </Nonce>
              <ChannelBindingToken>
                 <BYTES>74 6C 73 2D 75 6E 69 71
75 65 3A 20 87 E9 38 E3 2C 32 4F 6C F9 8E 5B </BYTES>
              </ChannelBindingToken>
              <Signature>
                 <BYTES>0D
40 B2 1B 57 0E AB FE EF 2F CD 58 DA 0F 7F BD 9C
D0 8B F2 15 85 F3 83 20 7C 22 F8 17 9A 95 8F </BYTES>
              </Signature>
            </SignedData>
          </FeatureData>
        </FeatureOpt>
        <TERMINATOR>
          <BYTE>FF </BYTE>
        </TERMINATOR>
      </FeatureExt>
    </Login7>
  </PacketData>
```

4.4 Login Response

Login response from the server to the client:

```
04 01 01 61 00 00 01 00 E3 1B 00 01 06 6D 00 61
00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
```


Copyright © 2014 Microsoft Corporation.

```
74 00 65 00 72 00 AB 58 00 45 16 00 00 02 00 25
00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
00 74 00 65 00 72 00 27 00 2E 00 00 00 00 00 00
00 E3 08 00 07 05 09 04 D0 00 34 00 E3 17 00 02
0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69
00 73 00 68 00 00 E3 13 00 04 04 34 00 30 00 39
00 36 00 04 34 00 30 00 39 00 36 00 AB 5C 00 47
16 00 00 01 00 27 00 43 00 68 00 61 00 6E 00 67
00 65 00 64 00 20 00 6C 00 61 00 6E 00 67 00 75
00 61 00 67 00 65 00 20 00 73 00 65 00 74 00 74
00 69 00 6E 00 67 00 20 00 74 00 6F 00 20 00 75
00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73
00 68 00 2E 00 00 00 00 00 00 00 AD 36 00 01 72
09 00 02 16 4D 00 69 00 63 00 72 00 6F 00 73 00
6F 00 66 00 74 00 20 00 53 00 51 00 4C 00 20 00
53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00
00 00 00 00 FD 00 00 00 00 00 00 00 00 00 00 00
00
```

```
  <PacketHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>01 </BYTE>
      <BYTE>61 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <TableResponse>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>1B 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>01 06 6D 00 61 00 73 00 74 00 65 00 72 00 06 6D 00 61 00
73 00 74 00 65 00 72 00 </BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <INFO>
```


Copyright © 2014 Microsoft Corporation.

```
          <TokenType>
            <BYTE>AB </BYTE>
          </TokenType>
          <Length>
            <USHORT>58 00 </USHORT>
          </Length>
          <Number>
            <LONG>45 16 00 00 </LONG>
          </Number>
          <State>
            <BYTE>02 </BYTE>
          </State>
          <Class>
            <BYTE>00 </BYTE>
          </Class>
          <MsgText>
            <US_UNICODE>
              <USHORTLEN>
                <USHORT>25 00 </USHORT>
              </USHORTLEN>
              <BYTES ascii="C.h.a.n.g.e.d. .d.a.t.a.b.a.s.e. .c.o.n.t.e.x.t.
.t.o. .'.m.a.s.t.e.r.'...">43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
64 00 61 00 74 00 61 00 62 00 61 00 73 00 65 00 20 00 63 00 6F 00 6E 00 74
00 65 00 78 00 74 00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73 00 74 00
65 00 72 00 27 00 2E 00 </BYTES>
            </US_UNICODE>
          </MsgText>
          <ServerName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>00 </BYTE>
              </BYTELEN>
              <BYTES ascii="">
              </BYTES>
            </B_UNICODE>
          </ServerName>
          <ProcName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>00 </BYTE>
              </BYTELEN>
              <BYTES ascii="">
              </BYTES>
            </B_UNICODE>
          </ProcName>
          <LineNumber>
            <LONG>00 00 00 00 </LONG>
          </LineNumber>
        </INFO>
        <ENVCHANGE>
          <TokenType>
            <BYTE>E3 </BYTE>
          </TokenType>
          <Length>
            <USHORT>08 00 </USHORT>
          </Length>
          <EnvChangeData>
            <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
          </EnvChangeData>
```


Copyright © 2014 Microsoft Corporation.

```
      </ENVCHANGE>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>17 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>02 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73
00 68 00 00 </BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <INFO>
        <TokenType>
          <BYTE>AB </BYTE>
        </TokenType>
        <Length>
          <USHORT>5C 00 </USHORT>
        </Length>
        <Number>
          <LONG>47 16 00 00 </LONG>
        </Number>
        <State>
          <BYTE>01 </BYTE>
        </State>
        <Class>
          <BYTE>00 </BYTE>
        </Class>
        <MsgText>
          <US_UNICODE>
            <USHORTLEN>
              <USHORT>27 00 </USHORT>
            </USHORTLEN>
            <BYTES ascii="C.h.a.n.g.e.d. .l.a.n.g.u.a.g.e. .s.e.t.t.i.n.g.
.t.o. .u.s._.e.n.g.l.i.s.h...">43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
00 6C 00 61 00 6E 00 67 00 75 00 61 00 67 00 65 00 20 00 73 00 65 00 74 00
74 00 69 00 6E 00 67 00 20 00 74 00 6F 00 20 00 75 00 73 00 5F 00 65 00 6E
00 67 00 6C 00 69 00 73 00 68 00 2E 00 </BYTES>
          </US_UNICODE>
        </MsgText>
        <ServerName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ServerName>
        <ProcName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ProcName>
```


Copyright © 2014 Microsoft Corporation.

```
        <LineNumber>
          <LONG>00 00 00 00 </LONG>
        </LineNumber>
      </INFO>
      <LOGINACK>
        <TokenType>
          <BYTE>AD </BYTE>
        </TokenType>
        <Length>
          <USHORT>36 00 </USHORT>
        </Length>
        <Interface>
          <BYTE>01 </BYTE>
        </Interface>
        <TDSVersion>
          <DWORD>72 09 00 02 </DWORD>
        </TDSVersion>
        <ProgName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>16 </BYTE>
            </BYTELEN>
            <BYTES ascii="M.i.c.r.o.s.o.f.t. .S.Q.L. .S.e.r.v.e.r.....">4D
00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00 20 00 53 00 51 00 4C 00
20 00 53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00 </BYTES>
          </B_UNICODE>
        </ProgName>
        <ProgVersion>
          <DWORD>00 00 00 00 </DWORD>
        </ProgVersion>
      </LOGINACK>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>13 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00
</BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>00 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>00 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
    </TableResponse>
```


Copyright © 2014 Microsoft Corporation.

```
</PacketData>
```

4.5 Login Response with Federated Authentication Feature Extension Acknowledgement

Login response from the server to the client, including the Feature Extension Acknowledgement token with a Federated Authentication feature extension option:

```
04 01 01 BC 01 4A 01 00 E3 1B 00 01 06 6D 00 61
00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
74 00 65 00 72 00 AB 62 00 45 16 00 00 02 00 25
00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
00 74 00 65 00 72 00 27 00 2E 00 05 63 00 6C 00
6F 00 75 00 64 00 00 01 00 00 00 E3 08 00 07 05
09 04 D0 00 34 00 E3 17 00 02 0A 75 00 73 00 5F
00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00
AB 66 00 47 16 00 00 01 00 27 00 43 00 68 00 61
00 6E 00 67 00 65 00 64 00 20 00 6C 00 61 00 6E
00 67 00 75 00 61 00 67 00 65 00 20 00 73 00 65
00 74 00 74 00 69 00 6E 00 67 00 20 00 74 00 6F
00 20 00 75 00 73 00 5F 00 65 00 6E 00 67 00 6C
00 69 00 73 00 68 00 2E 00 05 63 00 6C 00 6F 00
75 00 64 00 00 01 00 00 00 AD 36 00 01 74 00 00
04 16 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00
66 00 74 00 20 00 53 00 51 00 4C 00 20 00 53 00
65 00 72 00 76 00 65 00 72 00 00 00 00 00 00 0B 00
08 CB E3 13 00 04 04 34 00 30 00 39 00 36 00 04
34 00 30 00 39 00 36 00 AE 02 40 00 00 00 C9 08
46 4E 58 49 0C 71 80 72 CD 69 F0 EC 3D E2 F6 ED
75 8C 77 7D 9C B8 BB 87 4A 9C 90 80 A4 EE 40 B6
07 71 0E A8 3C 0E D7 DE 14 DE F3 8B 65 C2 06 8C
F9 51 D3 BC 32 55 15 A5 E4 A2 45 62 78 80 FF FD
00 00 00 00 00 00 00 00 00 00 00 00
```

```
<PacketHeader>
  <Type>
    <BYTE>04 </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>01 </BYTE>
    <BYTE>BC </BYTE>
  </Length>
  <SPID>
    <BYTE>01 </BYTE>
    <BYTE>4A </BYTE>
  </SPID>
  <Packet>
    <BYTE>01 </BYTE>
  </Packet>
  <Window>
    <BYTE>00 </BYTE>
```


Copyright © 2014 Microsoft Corporation.

```
      </Window>
    </PacketHeader>
    <PacketData>
      <TableResponse>
        <ENVCHANGE>
          <TokenType>
            <BYTE>E3 </BYTE>
          </TokenType>
          <Length>
            <USHORT>1B 00 </USHORT>
          </Length>
          <EnvChangeData>
            <BYTES>01 06 6D 00 61 00 73 00 74 00 65 00
  72 00 06 6D 00 61 00 73 00 74 00 65 00 72 00 </BYTES>
          </EnvChangeData>
        </ENVCHANGE>
        <INFO>
          <TokenType>
            <BYTE>AB </BYTE>
          </TokenType>
          <Length>
            <USHORT>62 00 </USHORT>
          </Length>
          <Number>
            <LONG>45 16 00 00 </LONG>
          </Number>
          <State>
            <BYTE>02 </BYTE>
          </State>
          <Class>
            <BYTE>00 </BYTE>
          </Class>
          <MsgText>
            <US_UNICODE>
              <USHORTLEN>
                <USHORT>25 00 </USHORT>
              </USHORTLEN>
              <BYTES ascii="C.h.a.n.g.e.d. .d.a.t.a.b.a.s.e. .c.o.n.t.e.x.t.
  .t.o. .'.m.a.s.t.e.r.'...">
  43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
  64 00 61 00 74 00 61 00 62 00 61 00 73 00 65 00
  20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74 00
  20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73 00
  74 00 65 00 72 00 27 00 2E 00 </BYTES>
            </US_UNICODE>
          </MsgText>
            <ServerName>
              <B_UNICODE>
                <BYTELEN>
                  <BYTE>05 </BYTE>
                </BYTELEN>
                <BYTES ascii="c.l.o.u.d.">
  63 00 6C 00 6F 00 75 00 64 00 </BYTES>
              </B_UNICODE>
            </ServerName>
            <ProcName>
              <B_UNICODE>
                <BYTELEN>
                  <BYTE>00 </BYTE>
```


Copyright © 2014 Microsoft Corporation.

```
              </BYTELEN>
              <BYTES ascii="">
              </BYTES>
            </B_UNICODE>
          </ProcName>
          <LineNumber>
            <LONG>01 00 00 00
          </LineNumber>
        </INFO>
        <ENVCHANGE>
          <TokenType>
            <BYTE>E3 </BYTE>
          </TokenType>
          <Length>
            <USHORT>08 00 </USHORT>
          </Length>
          <EnvChangeData>
            <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
          </EnvChangeData>
        </ENVCHANGE>
        <ENVCHANGE>
          <TokenType>
            <BYTE>E3 </BYTE>
          </TokenType>
          <Length>
            <USHORT>17 00 </USHORT>
          </Length>
          <EnvChangeData>
            <BYTES>02 0A 75 00 73 00 5F
00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00 </BYTES>
          </EnvChangeData>
        </ENVCHANGE>
        <INFO>
          <TokenType>
            <BYTE>AB </BYTE>
          </TokenType>
          <Length>
            <USHORT>66 00 </USHORT>
          </Length>
          <Number>
            <LONG>47 16 00 00 </LONG>
          </Number>
          <State>
            <BYTE>01 </BYTE>
          </State>
          <Class>
            <BYTE>00 </BYTE>
          </Class>
          <MsgText>
            <US_UNICODE>
              <USHORTLEN>
                <USHORT>27 00 </USHORT>
              </USHORTLEN>
              <BYTES ascii="C.h.a.n.g.e.d. .l.a.n.g.u.a.g.e. .s.e.t.t.i.n.g.
.t.o. .u.s._.e.n.g.l.i.s.h...">
43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
6C 00 61 00 6E 00 67 00 75 00 61 00 67 00 65 00
20 00 73 00 65 00 74 00 74 00 69 00 6E 00 67 00
20 00 74 00 6F 00 20 00 75 00 73 00 5F 00 65 00
```


Copyright © 2014 Microsoft Corporation.

```
6E 00 67 00 6C 00 69 00 73 00 68 00 2E 00 </BYTES>
            </US_UNICODE>
          </MsgText>
          <ServerName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>05 </Byte>
              </BYTELEN>
              <BYTES ascii="c.l.o.u.d.">
63 00 6C 00 6F 00 75 00 64 00 </BYTES>
            </B_UNICODE>
          </ServerName>
        <ProcName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ProcName>
        <LineNumber>
          <LONG>01 00 00 00 </LONG>
        </LineNumber>
      </INFO>
      <LOGINACK>
        <TokenType>
          <BYTE>AD </BYTE>
        </TokenType>
        <Length>
          <USHORT>36 00 </USHORT>
        </Length>
        <Interface>
          <BYTE>01 </BYTE>
        </Interface>
        <TDSVersion>
          <DWORD>74 00 00 04 </DWORD>
        </TDSVersion>
          <ProgName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>16 </BYTE>
              </BYTELEN>
              <BYTES ascii="M.i.c.r.o.s.o.f.t. .S.Q.L. .S.e.r.v.e.r.....">4D
00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00 20 00 53 00 51 00 4C 00
20 00 53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00 </BYTES>
          </B_UNICODE>
        </ProgName>
        <ProgVersion>
          <DWORD>0B 00 08 CB </DWORD>
        </ProgVersion>
      </LOGINACK>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>13 00 </USHORT>
        </Length>
```


Copyright © 2014 Microsoft Corporation.

```
        <EnvChangeData>
          <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00
</BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <FEATUREEXTACK>
        <TokenType>
          <BYTE>AE </BYTE>
        </TokenType>
        <FeatureAckOpt>
          <FeatureId>
            <BYTE>02 </BYTE>
          </FeatureId>
          <FeatureAckDataLen>
            <DWORD>40 00 00 00 </DWORD>
          </FeatureAckDataLen>
          <FeatureAckData>
            <Nonce>
              <BYTES>
C9 08 46 4E 58 49 0C 71 80 72 CD 69 F0 EC 3D E2
F6 ED 75 8C 77 7D 9C B8 BB 87 4A 9C 90 80 A4 EE </BYTES>
            </Nonce>
            <Signature>
              <BYTES>
40 B6 07 71 0E A8 3C 0E D7 DE 14 DE F3 8B 65 C2
06 8C F9 51 D3 BC 32 55 15 A5 E4 A2 45 62 78 80 </BYTES>
            </Signature>
          </FeatureAckData>
        </FeatureAckOpt>
        <FeatureAckOpt>
          <TERMINATOR>
            <BYTE>FF </BYTE>
          </TERMINATOR>
        </FeatureAckOpt>
      </FEATUREEXTACK>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>00 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>00 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
    </TableResponse>
  </PacketData>
```

4.6 SQL Batch Client Request

Client request sent from the client to the server:

Copyright © 2014 Microsoft Corporation.

```
01 01 00 5C 00 00 01 00 16 00 00 00 12 00 00 00
02 00 00 00 00 00 00 00 00 01 00 00 00 00 0A 00
73 00 65 00 6C 00 65 00 63 00 74 00 20 00 27 00
66 00 6F 00 6F 00 27 00 20 00 61 00 73 00 20 00
27 00 62 00 61 00 72 00 27 00 0A 00 20 00 20 00
20 00 20 00 20 00 20 00 20 00 20 00
```

```
  <PacketHeader>
    <Type>
      <BYTE>01 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>5C </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <SQLBatch>
      <All_HEADERS>
        <TotalLength>
          <DWORD>16 00 00 00 </DWORD>
        </TotalLength>
        <Header>
          <HeaderLength>
            <DWORD>12 00 00 00 </DWORD>
          </HeaderLength>
          <HeaderType>
            <USHORT>02 00 </USHORT>
          </HeaderType>
          <HeaderData>
            <MARS>
              <TransactionDescriptor>
                <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
              </TransactionDescriptor>
              <OutstandingRequestCount>
                <DWORD>00 00 00 00 </DWORD>
              </OutstandingRequestCount>
            </MARS>
          </HeaderData>
        </Header>
      </All_HEADERS>
      <SQLText>
        <UNICODESTREAM>
          <BYTES>0A 00 73 00 65 00 6C 00 65 00 63 00 74 00 20 00 27 00 66
00 6F 00 6F 00 27 00 20 00 61 00 73 00 20 00 27 00 62 00 61 00 72 00 27 00
0A 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00 </BYTES>
```


Copyright © 2014 Microsoft Corporation.

```
        </UNICODESTREAM>
      </SQLText>
    </SQLBatch>
  </PacketData>
```

4.7 SQL Batch Server Response

Server response sent from the server to the client:

```
04 01 00 33 00 00 01 00 81 01 00 00 00 00 00 20
00 A7 03 00 09 04 D0 00 34 03 62 00 61 00 72 00
D1 03 00 66 6F 6F FD 10 00 C1 00 01 00 00 00 00
00 00 00

  <PacketHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>33 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <TableResponse>
      <COLMETADATA>
        <TokenType>
          <BYTE>81 </BYTE>
        </TokenType>
        <Count>
          <USHORT>01 00 </USHORT>
        </Count>
        <ColumnData>
          <UserType>
            <ULONG>00 00 00 00 </ULONG>
          </UserType>
          <Flags>
            <USHORT>20 00 </USHORT>
          </Flags>
          <TYPE_INFO>
            <VARLENTYPE>
              <USHORTLEN_TYPE>
                <BYTE>A7 </BYTE>
              </USHORTLEN_TYPE>
```


Copyright © 2014 Microsoft Corporation.

```
            </VARLENTYPE>
            <TYPE_VARLEN>
              <USHORTCHARBINLEN>
                <USHORT>03 00 </USHORT>
              </USHORTCHARBINLEN>
            </TYPE_VARLEN>
            <COLLATION>
              <BYTES>09 04 D0 00 34 </BYTES>
            </COLLATION>
          </TYPE_INFO>
          <ColName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>03 </BYTE>
              </BYTELEN>
              <BYTES ascii="b.a.r.">62 00 61 00 72 00 </BYTES>
            </B_UNICODE>
          </ColName>
        </ColumnData>
      </COLMETADATA>
      <ROW>
        <TokenType>
          <BYTE>D1 </BYTE>
        </TokenType>
        <TYPE_VARBYTE>
          <TYPE_VARLEN>
            <USHORTCHARBINLEN>
              <USHORT>03 00 </USHORT>
            </USHORTCHARBINLEN>
          </TYPE_VARLEN>
          <BYTES ascii="foo">66 6F 6F </BYTES>
        </TYPE_VARBYTE>
      </ROW>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>10 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>C1 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
    </TableResponse>
  </PacketData>
```

4.8 RPC Client Request

RPC request sent from the client to the server:

```
03 01 00 2F 00 00 01 00 16 00 00 00 12 00 00 00
02 00 00 00 00 00 00 00 00 01 00 00 00 00 04 00
66 00 6F 00 6F 00 33 00 00 00 00 02 26 02 00
```


Copyright © 2014 Microsoft Corporation.

```
<PacketHeader>
  <Type>
    <BYTE>03 </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>00 </BYTE>
    <BYTE>2F </BYTE>
  </Length>
  <SPID>
    <BYTE>00 </BYTE>
    <BYTE>00 </BYTE>
  </SPID>
  <Packet>
    <BYTE>01 </BYTE>
  </Packet>
  <Window>
    <BYTE>00 </BYTE>
  </Window>
</PacketHeader>
<PacketData>
  <RPCRequest>
    <All_HEADERS>
      <TotalLength>
        <DWORD>16 00 00 00 </DWORD>
      </TotalLength>
      <Header>
        <HeaderLength>
          <DWORD>12 00 00 00 </DWORD>
        </HeaderLength>
        <HeaderType>
          <USHORT>02 00 </USHORT>
        </HeaderType>
        <HeaderData>
          <MARS>
            <TransactionDescriptor>
              <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
            </TransactionDescriptor>
            <OutstandingRequestCount>
              <DWORD>00 00 00 00 </DWORD>
            </OutstandingRequestCount>
          </MARS>
        </HeaderData>
      </Header>
    </All_HEADERS>
    <RPCReqBatch>
      <NameLenProcID>
        <ProcName>
          <US_UNICODE>
            <USHORTLEN>
              <USHORT>04 00 </USHORT>
            </USHORTLEN>
            <BYTES ascii="f.o.o.3.">66 00 6F 00 6F 00 33 00 </BYTES>
          </US_UNICODE>
        </ProcName>
      </NameLenProcID>
```


Copyright © 2014 Microsoft Corporation.

```
        <OptionFlags>
          <fWithRecomp>
            <BIT>0</BIT>
          </fWithRecomp>
          <fNoMetaData>
            <BIT>0</BIT>
          </fNoMetaData>
          <fReuseMetaData>
            <BIT>false</BIT>
          </fReuseMetaData>
        </OptionFlags>
        <ParameterData>
          <ParamMetaData>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>00 </BYTE>
              </BYTELEN>
              <BYTES ascii="">
              </BYTES>
            </B_UNICODE>
            <StatusFlags>
              <fByRefValue>
                <BIT>0</BIT>
              </fByRefValue>
              <fDefaultValue>
                <BIT>1</BIT>
              </fDefaultValue>
            </StatusFlags>
            <TYPE_INFO>
              <VARLENTYPE>
                <BYTELEN_TYPE>
                  <BYTE>26 </BYTE>
                </BYTELEN_TYPE>
              </VARLENTYPE>
              <TYPE_VARLEN>
                <BYTELEN>
                  <BYTE>02 </BYTE>
                </BYTELEN>
              </TYPE_VARLEN>
            </TYPE_INFO>
          </ParamMetaData>
          <ParamLenData>
            <TYPE_VARBYTE>
              <TYPE_VARLEN>
                <BYTELEN>
                  <BYTE>00 </BYTE>
                </BYTELEN>
              </TYPE_VARLEN>
              <BYTES>
              </BYTES>
            </TYPE_VARBYTE>
          </ParamLenData>
        </ParameterData>
      </RPCReqBatch>
    </RPCRequest>
  </PacketData>
```


Copyright © 2014 Microsoft Corporation.

4.9 RPC Server Response

RPC response sent from the server to the client:

```
04 01 00 27 00 00 01 00 FF 11 00 C1 00 01 00 00
00 00 00 00 00 79 00 00 00 00 FE 00 00 E0 00 00
00 00 00 00 00 00 00
```

```
  <PacketHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>27 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <TableResponse>
      <DONEINPROC>
        <TokenType>
          <BYTE>FF </BYTE>
        </TokenType>
        <Status>
          <USHORT>11 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>C1 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONEINPROC>
      <RETURNSTATUS>
        <TokenType>
          <BYTE>79 </BYTE>
        </TokenType>
        <VALUE>
          <LONG>00 00 00 00 </LONG>
        </VALUE>
      </RETURNSTATUS>
      <DONEPROC>
        <TokenType>
          <BYTE>FE </BYTE>
        </TokenType>
        <Status>
```


Copyright © 2014 Microsoft Corporation.

```
          <USHORT>00 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>E0 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONEPROC>
    </TableResponse>
  </PacketData>
```

4.10 Attention Request

Attention request sent from client to server:

```
06 01 00 08 00 00 01 00

<PacketHeader>
  <Type>
    <BYTE>06</BYTE>
  </Type>
  <Status>
    <BYTE>01</BYTE>
  </Status>
  <Length>
    <BYTE>00</BYTE>
    <BYTE>08</BYTE>
  </Length>
  <SPID>
    <BYTE>00</BYTE>
    <BYTE>00</BYTE>
  </SPID>
  <Packet>
    <BYTE>01</BYTE>
  </Packet>
  <Window>
    <BYTE>00</BYTE>
  </Window>
</PacketHeader>
```

4.11 SSPI Message

SSPI message carrying SSPI payload sent from client to server:

```
11 01 00 60 00 00 01 00 4E 54 4C 4D 53 53 50 00
03 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
58 00 00 00 15 C2 88 E2 06 00 71 17 00 00 00 0F
30 81 C1 7D 59 5F E9 3E 1A 7C 98 05 01 72 5C 4F


<PacketHeader>
```


Copyright © 2014 Microsoft Corporation.

```
  <Type>
    <BYTE>11 </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>00 </BYTE>
    <BYTE>60 </BYTE>
  </Length>
  <SPID>
    <BYTE>00 </BYTE>
    <BYTE>00 </BYTE>
  </SPID>
  <Packet>
    <BYTE>01 </BYTE>
  </Packet>
  <Window>
    <BYTE>00 </BYTE>
  </Window>
</PacketHeader>
<PacketData>
  <SSPI>
    <BYTES>4E 54 4C 4D 53 53 50 00 03 00 00 00 00 00 00 00 58 00 00 00 00
00 00 00 58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00 58 00 00 00 00
00 00 58 00 00 00 00 00 00 00 58 00 00 00 15 C2 88 E2 06 00 71 17 00 00
0F 30 81 C1 7D 59 5F E9 3E 1A 7C 98 05 01 72 5C 4F </BYTES>
  </SSPI>
</PacketData>
```

4.12 SQL Command with Binary Data

BULKLOADBCP request sent from client to server:

```
07 01 00 26 00 00 01 00 81 01 00 00 00 00 00 05
00 32 02 63 00 31 00 D1 00 FD 00 00 00 00 00 00
00 00 00 00 00 00
```

```
<PacketHeader>
  <Type>
    <BYTE>07 </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>00 </BYTE>
    <BYTE>26 </BYTE>
  </Length>
  <SPID>
    <BYTE>00 </BYTE>
    <BYTE>00 </BYTE>
  </SPID>
  <Packet>
```


Copyright © 2014 Microsoft Corporation.

```
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </PacketHeader>
  <PacketData>
    <BulkLoadBCP>
      <COLMETADATA>
        <TokenType>
          <BYTE>81 </BYTE>
        </TokenType>
        <Count>
          <USHORT>01 00 </USHORT>
        </Count>
        <ColumnData>
          <UserType>
            <ULONG>00 00 00 00 </ULONG>
          </UserType>
          <Flags>
            <USHORT>05 00 </USHORT>
          </Flags>
          <TYPE_INFO>
            <FIXEDLENTYPE>
              <BYTE>32 </BYTE>
            </FIXEDLENTYPE>
          </TYPE_INFO>
          <ColName>
            <B_UNICODE>
              <BYTELEN>
                <BYTE>02 </BYTE>
              </BYTELEN>
              <BYTES ascii="c.1.">63 00 31 00 </BYTES>
            </B_UNICODE>
          </ColName>
        </ColumnData>
      </COLMETADATA>
      <ROW>
        <TokenType>
          <BYTE>D1 </BYTE>
        </TokenType>
        <TYPE_VARBYTE>
          <BYTES>00 </BYTES>
        </TYPE_VARBYTE>
      </ROW>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>00 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>00 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
```


Copyright © 2014 Microsoft Corporation.

```
  </BulkLoadBCP>
</PacketData>
```

4.13 Transaction Manager Request

Transaction Manager Request sent from client to server:

```
0E 01 00 20 00 00 01 00 16 00 00 00
12 00 00 00 02 00 00 00 00 00 00 00
00 00 00 00 00 01 06 00


<PacketHeader>
  <Type>
    <BYTE>0E </BYTE>
  </Type>
  <Status>
    <BYTE>01 </BYTE>
  </Status>
  <Length>
    <BYTE>00 </BYTE>
    <BYTE>20 </BYTE>
  </Length>
  <SPID>
    <BYTE>00 </BYTE>
    <BYTE>00 </BYTE>
  </SPID>
  <Packet>
    <BYTE>01 </BYTE>
  </Packet>
  <Window>
    <BYTE>00 </BYTE>
  </Window>
</PacketHeader>
<PacketData>
  <TransMgrReq>
    <All_HEADERS>
      <TotalLength>
        <DWORD>16 00 00 00 </DWORD>
      </TotalLength>
      <Header>
        <HeaderLength>
          <DWORD>12 00 00 00 </DWORD>
        </HeaderLength>
        <HeaderType>
          <USHORT>02 00 </USHORT>
        </HeaderType>
        <HeaderData>
          <MARS>
            <TransactionDescriptor>
              <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
            </TransactionDescriptor>
            <OutstandingRequestCount>
              <DWORD>00 00 00 00 </DWORD>
            </OutstandingRequestCount>
```


Copyright © 2014 Microsoft Corporation.

```
          </MARS>
        </HeaderData>
      </Header>
    </All_HEADERS>
    <RequestType>
      <USHORT>16 00 </USHORT>
    </RequestType>
    <RequestPayload>
      <TM_PROMOTE_XACT>
      </TM_PROMOTE_XACT>
    </RequestPayload>
  </TransMgrReq>
</PacketData>
```

4.14 TVP Insert Statement

TVP insert statement sent from client to server:

```
03 01 00 52 00 00 01 00 16 00 00 00
12 00 00 00 02 00 00 00 00 00 00 00
00 00 00 00 00 01 03 00 66 00 6F 00
6F 00 00 00 00 00 F3 00 03 64 00 62
00 6F 00 07 74 00 76 00 70 00 74 00
79 00 70 00 65 00 01 00 00 00 00 00
00 00 26 01 00 00 01 01 02 00
```

```
<tds version="katmai">
  <BufferData>
    <RPCRequest>
      <RPCReqBatch>
        <NameLenProcID>
          <ProcName>
            <US_UNICODE>
              <USHORTLEN>
                <USHORT>03 00 </USHORT>
              </USHORTLEN>
              <BYTES ascii="f.o.o.">66 00 6F 00 6F 00 </BYTES>
            </US_UNICODE>
          </ProcName>
        </NameLenProcID>
        <OptionFlags>
          <fWithRecomp>
            <BIT>false</BIT>
          </fWithRecomp>
          <fNoMetaData>
            <BIT>false</BIT>
          </fNoMetaData>
          <fReuseMetaData>
            <BIT>false</BIT>
          </fResuseMetaData>
        </OptionFlags>
        <ParameterData>
          <ParamMetaData>
```


Copyright © 2014 Microsoft Corporation.

```
<B_UNICODE>
  <BYTELEN>
    <BYTE>00 </BYTE>
  </BYTELEN>
  <BYTES ascii="">
  </BYTES>
</B_UNICODE>
<StatusFlags>
  <fByRefValue>
    <BIT>false</BIT>
  </fByRefValue>
  <fDefaultValue>
    <BIT>false</BIT>
  </fDefaultValue>
  <fCookie>
    <BIT>false</BIT>
  </fCookie>
</StatusFlags>
<TYPE_INFO>
  <TVP_TYPE_INFO>
    <TVP_TYPE>
      <BYTE>F3 </BYTE>
    </TVP_TYPE>
    <TVP_TYPE_NAME>
      <DbName>
        <B_UNICODE></B_UNICODE>
      </DbName>
      <OwningSchema>
        <B_UNICODE>dbo</B_UNICODE>
      </OwningSchema>
      <TypeName>
        <B_UNICODE>tvptype</B_UNICODE>
      </TypeName>
    </TVP_TYPE_NAME>
    <TVP_COLMETADATA>
      <Count>
        <USHORT>01 00 </USHORT>
      </Count>
      <TvpColumnMetaData>
        <UserType>
          <ULONG>00 00 00 00 </ULONG>
        </UserType>
        <Flags>
          <USHORT>00 00 </USHORT>
        </Flags>
<TYPE_INFO>
  <VARLENTYPE>
    <BYTELEN_TYPE>
      <BYTE>26 </BYTE>
    </BYTELEN_TYPE>
  </VARLENTYPE>
  <TYPE_VARLEN>
    <BYTELEN>
      <BYTE>01 </BYTE>
    </BYTELEN>
  </TYPE_VARLEN>
</TYPE_INFO>
        <ColName>
          <B_UNICODE>
```


Copyright © 2014 Microsoft Corporation.

```
                        <BYTELEN>
                          <BYTE>00 </BYTE>
                        </BYTELEN>
                        <BYTES ascii="">
                        </BYTES>
                      </B_UNICODE>
                    </ColName>
                  </TvpColumnMetaData>
                </TVP_COLMETADATA>
                <TVP_END_TOKEN>
                  <TokenType>
                    <BYTE>00 </BYTE>
                  </TokenType>
                </TVP_END_TOKEN>
                <TVP_ROW>
                  <TokenType>
                    <BYTE>01 </BYTE>
                  </TokenType>
                  <AllColumnData>
              <TYPE_VARBYTE>
                <TYPE_VARLEN>
                  <BYTELEN>
                    <BYTE>01</BYTE>
                  </BYTELEN>
                 <BYTES>02</BYTES>
                </TYPE_VARLEN>
              </TYPE_VARBYTE>
                  </AllColumnData>
                </TVP_ROW>
                <TVP_END_TOKEN>
                  <TokenType>
                    <BYTE>00 </BYTE>
                  </TokenType>
                </TVP_END_TOKEN>
              </TVP_TYPE_INFO>
            </TYPE_INFO>
          </ParamMetaData>
          <ParamLenData>
          </ParamLenData>
        </ParameterData>
      </RPCReqBatch>
    </RPCRequest>
  </BufferData>
</tds>
```

4.15 SparseColumn Select Statement

SparseColumn select statement sent from client to server:

```
04 01 01 B9 00 00 01 00 81 02 00 00 00 00 00 09 00
26 04 02 69 00 64 00 00 00 00 00 0B 04 F1 00 11 73
00 70 00 61 00 72 00 73 00 65 00 50 00 72 00 6F 00
70 00 65 00 72 00 74 00 79 00 53 00 65 00 74 00 D1
04 01 00 00 00 FE FF FF FF FF FF FF FF 7A 00 00 00
3C 00 73 00 70 00 61 00 72 00 73 00 65 00 50 00 72
```


Copyright © 2014 Microsoft Corporation.

```
00 6F 00 70 00 31 00 3E 00 31 00 30 00 30 00 30 00
3C 00 2F 00 73 00 70 00 61 00 72 00 73 00 65 00 50
00 72 00 6F 00 70 00 31 00 3E 00 3C 00 73 00 70 00
61 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
00 3E 00 66 00 6F 00 6F 00 3C 00 2F 00 73 00 70 00
61 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
00 3E 00 00 00 00 00 D1 04 02 00 00 00 FE FF FF FF
FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61 00 72
00 73 00 65 00 50 00 72 00 6F 00 70 00 31 00 3E 00
31 00 30 00 30 00 30 00 3C 00 2F 00 73 00 70 00 61
00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 31
00 3E 00 00 00 00 00 D1 04 03 00 00 00 FE FF FF
FF FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61
00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
00 3E 00 61 00 62 00 63 00 64 00 3C 00 2F 00 73
00 70 00 61 00 72 00 73 00 65 00 50 00 72 00 6F
00 70 00 32 00 3E 00 00 00 00 00 FD 10 00 C1 00
0A 00 00 00 00 00 00 00
```

```
<tds version="katmai">
  <BufferHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>01 </BYTE>
      <BYTE>B9 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </BufferHeader>
  <BufferData>
    <TableResponse>
      <COLMETADATA>
        <TokenType>
          <BYTE>81 </BYTE>
        </TokenType>
        <Count>
          <USHORT>02 00 </USHORT>
        </Count>
        <ColumnData>
          <UserType>
            <ULONG>00 00 00 00 </ULONG>
          </UserType>
          <Flags>
            <USHORT>09 00 </USHORT>
```


Copyright © 2014 Microsoft Corporation.

```
            </Flags>
            <TYPE_INFO>
              <VARLENTYPE>
                <BYTELEN_TYPE>
                  <BYTE>26 </BYTE>
                </BYTELEN_TYPE>
              </VARLENTYPE>
              <TYPE_VARLEN>
                <BYTELEN>
                  <BYTE>04 </BYTE>
                </BYTELEN>
              </TYPE_VARLEN>
            </TYPE_INFO>
            <ColName>
              <B_UNICODE>
                <BYTELEN>
                  <BYTE>02 </BYTE>
                </BYTELEN>
                <BYTES ascii="i.d.">69 00 64 00 </BYTES>
              </B_UNICODE>
            </ColName>
          </ColumnData>
          <ColumnData>
            <UserType>
              <ULONG>00 00 00 00 </ULONG>
            </UserType>
            <Flags fSparseColumn="true">
              <USHORT>0B 04 </USHORT>
            </Flags>
            <TYPE_INFO>
              <VARLENTYPE>
                <USHORTLEN_TYPE>
                  <BYTE>F1 </BYTE>
                </USHORTLEN_TYPE>
              </VARLENTYPE>
              <XML_INFO>
                <SCHEMA_PRESENT>
                  <BYTE>00 </BYTE>
                </SCHEMA_PRESENT>
              </XML_INFO>
            </TYPE_INFO>
            <ColName>
              <B_UNICODE>
                <BYTELEN>
                  <BYTE>11 </BYTE>
                </BYTELEN>
                <BYTES ascii="s.p.a.r.s.e.P.r.o.p.e.r.t.y.S.e.t.">73 00 70 00 61 00 72 00 73 00
65 00 50 00 72 00 6F 00 70 00 65 00 72 00 74 00 79 00 53 00 65 00 74 00 </BYTES>
              </B_UNICODE>
            </ColName>
          </ColumnData>
        </COLMETADATA>
        <ROW>
          <TokenType>
            <BYTE>D1 </BYTE>
          </TokenType>
          <TYPE_VARBYTE>
            <TYPE_VARLEN>
              <BYTELEN>
```


Copyright © 2014 Microsoft Corporation.

```
              <BYTE>04 </BYTE>
            </BYTELEN>
          </TYPE_VARLEN>
          <BYTES>01 00 00 00 </BYTES>
        </TYPE_VARBYTE>
        <TYPE_VARBYTE>
          <BYTES>FE FF FF FF FF FF FF FF 7A 00 00 00 3C 00 73 00 70 00 61 00 72 00 73 00 65
00 50 00 72 00 6F 00 70 00 31 00 3E 00 31 00 30 00 30 00 30 00 3C 00 2F 00 73 00 70 00 61 00
72 00 73 00 65 00 50 00 72 00 6F 00 70 00 31 00 3E 00 3C 00 73 00 70 00 61 00 72 00 73 00 65
00 50 00 72 00 6F 00 70 00 32 00 3E 00 66 00 6F 00 6F 00 3C 00 2F 00 73 00 70 00 61 00 72 00
73 00 65 00 50 00 72 00 6F 00 70 00 32 00 3E 00 00 00 00 00 </BYTES>
        </TYPE_VARBYTE>
      </ROW>
      <ROW>
        <TokenType>
          <BYTE>D1 </BYTE>
        </TokenType>
        <TYPE_VARBYTE>
          <TYPE_VARLEN>
            <BYTELEN>
              <BYTE>04 </BYTE>
            </BYTELEN>
          </TYPE_VARLEN>
          <BYTES>02 00 00 00 </BYTES>
        </TYPE_VARBYTE>
        <TYPE_VARBYTE>
          <BYTES>FE FF FF FF FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61 00 72 00 73 00 65
00 50 00 72 00 6F 00 70 00 31 00 3E 00 31 00 30 00 30 00 30 00 3C 00 2F 00 73 00 70 00 61 00
72 00 73 00 65 00 50 00 72 00 6F 00 70 00 31 00 3E 00 00 00 00 00 </BYTES>
        </TYPE_VARBYTE>
      </ROW>
      <ROW>
        <TokenType>
          <BYTE>D1 </BYTE>
        </TokenType>
        <TYPE_VARBYTE>
          <TYPE_VARLEN>
            <BYTELEN>
              <BYTE>04 </BYTE>
            </BYTELEN>
          </TYPE_VARLEN>
          <BYTES>03 00 00 00 </BYTES>
        </TYPE_VARBYTE>
        <TYPE_VARBYTE>
          <BYTES>FE FF FF FF FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61 00 72 00 73 00 65
00 50 00 72 00 6F 00 70 00 32 00 3E 00 61 00 62 00 63 00 64 00 3C 00 2F 00 73 00 70 00 61 00
72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32 00 3E 00 00 00 00 00 </BYTES>
        </TYPE_VARBYTE>
      </ROW>
      <NBCROW>
        <TokenType>
          <BYTE>D2 </BYTE>
        </TokenType>
        <NBCBitMap>
          <BYTES>02 </BYTES>
        </NBCBitMap>
        <TYPE_VARBYTE>
          <TYPE_VARLEN>
            <BYTELEN>
              <BYTE>04 </BYTE>
```


Copyright © 2014 Microsoft Corporation.

```
      </BYTELEN>
    </TYPE_VARLEN>
    <BYTES>04 00 00 00 </BYTES>
  </TYPE_VARBYTE>
</NBCROW>
<NBCROW>
  <TokenType>
    <BYTE>D2 </BYTE>
  </TokenType>
  <NBCBitMap>
    <BYTES>02 </BYTES>
  </NBCBitMap>
  <TYPE_VARBYTE>
    <TYPE_VARLEN>
      <BYTELEN>
        <BYTE>04 </BYTE>
      </BYTELEN>
    </TYPE_VARLEN>
    <BYTES>05 00 00 00 </BYTES>
  </TYPE_VARBYTE>
</NBCROW>
<NBCROW>
  <TokenType>
    <BYTE>D2 </BYTE>
  </TokenType>
  <NBCBitMap>
    <BYTES>02 </BYTES>
  </NBCBitMap>
  <TYPE_VARBYTE>
    <TYPE_VARLEN>
      <BYTELEN>
        <BYTE>04 </BYTE>
      </BYTELEN>
    </TYPE_VARLEN>
    <BYTES>06 00 00 00 </BYTES>
  </TYPE_VARBYTE>
</NBCROW>
<NBCROW>
  <TokenType>
    <BYTE>D2 </BYTE>
  </TokenType>
  <NBCBitMap>
    <BYTES>02 </BYTES>
  </NBCBitMap>
  <TYPE_VARBYTE>
    <TYPE_VARLEN>
      <BYTELEN>
        <BYTE>04 </BYTE>
      </BYTELEN>
    </TYPE_VARLEN>
    <BYTES>07 00 00 00 </BYTES>
  </TYPE_VARBYTE>
</NBCROW>
<NBCROW>
  <TokenType>
    <BYTE>D2 </BYTE>
  </TokenType>
  <NBCBitMap>
    <BYTES>02 </BYTES>
```


Copyright © 2014 Microsoft Corporation.

```
          </NBCBitMap>
          <TYPE_VARBYTE>
            <TYPE_VARLEN>
              <BYTELEN>
                <BYTE>04 </BYTE>
              </BYTELEN>
            </TYPE_VARLEN>
            <BYTES>08 00 00 00 </BYTES>
          </TYPE_VARBYTE>
        </NBCROW>
        <NBCROW>
          <TokenType>
            <BYTE>D2 </BYTE>
          </TokenType>
          <NBCBitMap>
            <BYTES>02 </BYTES>
          </NBCBitMap>
          <TYPE_VARBYTE>
            <TYPE_VARLEN>
              <BYTELEN>
                <BYTE>04 </BYTE>
              </BYTELEN>
            </TYPE_VARLEN>
            <BYTES>09 00 00 00 </BYTES>
          </TYPE_VARBYTE>
        </NBCROW>
        <NBCROW>
          <TokenType>
            <BYTE>D2 </BYTE>
          </TokenType>
          <NBCBitMap>
            <BYTES>02 </BYTES>
          </NBCBitMap>
          <TYPE_VARBYTE>
            <TYPE_VARLEN>
              <BYTELEN>
                <BYTE>04 </BYTE>
              </BYTELEN>
            </TYPE_VARLEN>
            <BYTES>0A 00 00 00 </BYTES>
          </TYPE_VARBYTE>
        </NBCROW>
        <DONE>
          <TokenType>
            <BYTE>FD </BYTE>
          </TokenType>
          <Status>
            <USHORT>10 00 </USHORT>
          </Status>
          <CurCmd>
            <USHORT>C1 00 </USHORT>
          </CurCmd>
          <DoneRowCount>
            <LONGLONG>0A 00 00 00 00 00 00 00 </LONGLONG>
          </DoneRowCount>
        </DONE>
      </TableResponse>
    </BufferData>
  </tds>
```


Copyright © 2014 Microsoft Corporation.

4.16 FeatureExt with SessionRecovery Feature Data

A login message that contains FeatureExt data for the SessionRecovery feature:

```
10 01 01 0D 00 00 01 00 05 01 00 00 04 00 00 74
00 10 00 00 00 00 00 07 00 01 00 00 00 00 00 00
E0 03 00 10 E0 01 00 00 09 04 00 00 5E 00 00 00
5E 00 02 00 62 00 08 00 72 00 07 00 80 00 00 00
80 00 04 00 84 00 04 00 8C 00 00 00 8C 00 06 00
00 50 8B E2 B7 8F 98 00 00 00 98 00 00 00 98 00
00 00 00 00 00 00 00 73 00 61 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 4F 00 53 00 51 00
4C 00 2D 00 33 00 32 00 98 00 00 00 4F 00 44 00
42 00 43 00 74 00 65 00 6D 00 70 00 64 00 62 00
01 67 00 00 00 56 00 00 00 06 6D 00 61 00 73 00
74 00 65 00 72 00 05 09 04 D0 00 34 0A 75 00 73
00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68
00 00 09 00 60 81 14 FF E7 FF FF 00 02 02 07 01
04 01 00 05 04 FF FF FF FF 06 01 00 07 01 02 08
08 00 00 00 00 00 00 00 00 09 04 FF FF FF FF 09
00 00 00 00 00 00 09 04 28 23 00 00 FF
```

```
<tds version="latest">
  <BufferHeader>
    <Type>
      <BYTE>10 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>01 </BYTE>
      <BYTE>0D </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </BufferHeader>
  <BufferData>
    <Login7>
      <Length>
        <DWORD>05 01 00 00 </DWORD>
      </Length>
      <TDSVersion>
        <DWORD>04 00 00 74 </DWORD>
      </TDSVersion>
```


Copyright © 2014 Microsoft Corporation.

```
<PacketSize>
  <DWORD>00 10 00 00 </DWORD>
</PacketSize>
<ClientProgVer>
  <DWORD>00 00 00 07 </DWORD>
</ClientProgVer>
<ClientPID>
  <DWORD>00 01 00 00 </DWORD>
</ClientPID>
<ConnectionID>
  <DWORD>00 00 00 00 </DWORD>
</ConnectionID>
<OptionFlags1>
  <BYTE>E0 </BYTE>
</OptionFlags1>
<OptionFlags2>
  <BYTE>03 </BYTE>
</OptionFlags2>
<TypeFlags>
  <BYTE>00 </BYTE>
</TypeFlags>
<OptionFlags3>
  <BYTE>10 </BYTE>
</OptionFlags3>
<ClientTimZone>
  <DWORD>E0 01 00 00 </DWORD>
</ClientTimZone>
<ClientLCID>
  <DWORD>09 04 00 00 </DWORD>
</ClientLCID>
<OffsetLength>
  <ibHostName>
    <USHORT>5E 00 </USHORT>
  </ibHostName>
  <cchHostName>
    <USHORT>00 00 </USHORT>
  </cchHostName>
  <ibUserName>
    <USHORT>5E 00 </USHORT>
  </ibUserName>
  <cchUserName>
    <USHORT>02 00 </USHORT>
  </cchUserName>
  <ibPassword>
    <USHORT>62 00 </USHORT>
  </ibPassword>
  <cchPassword>
    <USHORT>08 00 </USHORT>
  </cchPassword>
  <ibAppName>
    <USHORT>72 00 </USHORT>
  </ibAppName>
  <cchAppName>
    <USHORT>07 00 </USHORT>
  </cchAppName>
  <ibServerName>
    <USHORT>80 00 </USHORT>
  </ibServerName>
  <cchServerName>
```


Copyright © 2014 Microsoft Corporation.

```
          <USHORT>00 00 </USHORT>
        </cchServerName>
        <ibExtension>
          <USHORT>80 00 </USHORT>
        </ibExtension>
        <cbExtension>
          <USHORT>04 00 </USHORT>
        </cbExtension>
        <ibCltIntName>
          <USHORT>84 00 </USHORT>
        </ibCltIntName>
        <cchCltIntName>
          <USHORT>04 00 </USHORT>
        </cchCltIntName>
        <ibLanguage>
          <USHORT>8C 00 </USHORT>
        </ibLanguage>
        <cchLanguage>
          <USHORT>00 00 </USHORT>
        </cchLanguage>
        <ibDatabase>
          <USHORT>8C 00 </USHORT>
        </ibDatabase>
        <cchDatabase>
          <USHORT>06 00 </USHORT>
        </cchDatabase>
        <ClientID>
          <BYTES>00 50 8B E2 B7 8F </BYTES>
        </ClientID>
        <ibSSPI>
          <USHORT>98 00 </USHORT>
        </ibSSPI>
        <cbSSPI>
          <USHORT>00 00 </USHORT>
        </cbSSPI>
        <ibAtchDBFile>
          <USHORT>98 00 </USHORT>
        </ibAtchDBFile>
        <cchAtchDBFile>
          <USHORT>00 00 </USHORT>
        </cchAtchDBFile>
        <ibChangePassword>
          <USHORT>98 00 </USHORT>
        </ibChangePassword>
        <cchChangePassword>
          <USHORT>00 00 </USHORT>
        </cchChangePassword>
        <cbSSPILong>
          <LONG>00 00 00 00 </LONG>
        </cbSSPILong>
      </OffsetLength>
      <Data>
        <BYTES>73 00 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4F 00 53 00 51 00
4C 00 2D 00 33 00 32 00 98 00 00 00 4F 00 44 00 42 00 43 00 74 00 65 00 6D 00 70 00 64 00 62
00 </BYTES>
      </Data>
      <FeatureExt>
        <FeatureOpt>
          <FeatureId>
```


Copyright © 2014 Microsoft Corporation.

```
            <BYTE>01 </BYTE>
          </FeatureId>
          <FeatureDataLen>
            <DWORD>67 00 00 00 </DWORD>
          </FeatureDataLen>
          <FeatureData>
            <InitSessionRecoveryData>
              <Length>
                <DWORD>56 00 00 00 </DWORD>
              </Length>
              <RecoveryDatabase>
                <B_VARCHAR>
                  <BYTE>06 </BYTE>
                  <BYTES ascii="m.a.s.t.e.r.">6D 00 61 00 73 00 74 00 65 00 72 00 </BYTES>
                </B_VARCHAR>
              </RecoveryDatabase>
              <RecoveryCollation>
                <BYTELEN>
                  <BYTE>05 </BYTE>
                </BYTELEN>
                <BYTES>09 04 D0 00 34 </BYTES>
              </RecoveryCollation>
              <RecoveryLanguage>
                <B_VARCHAR>
                  <BYTE>0A </BYTE>
                  <BYTES ascii="u.s._.e.n.g.l.i.s.h.">75 00 73 00 5F 00 65 00 6E 00 67 00 6C
00 69 00 73 00 68 00 </BYTES>
                </B_VARCHAR>
              </RecoveryLanguage>
              <SessionStateDataSet>
                <SessionStateData>
                  <StateId>
                    <BYTE>00 </BYTE>
                  </StateId>
                  <StateLen>
                    <BYTE>09 </BYTE>
                  </StateLen>
                  <StateValue>
                    <BYTES>00 60 81 14 FF E7 FF FF 00 </BYTES>
                  </StateValue>
                </SessionStateData>
                <SessionStateData>
                  <StateId>
                    <BYTE>02 </BYTE>
                  </StateId>
                  <StateLen>
                    <BYTE>02 </BYTE>
                  </StateLen>
                  <StateValue>
                    <BYTES>07 01 </BYTES>
                  </StateValue>
                </SessionStateData>
                <SessionStateData>
                  <StateId>
                    <BYTE>04 </BYTE>
                  </StateId>
                  <StateLen>
                    <BYTE>01 </BYTE>
                  </StateLen>
```


Copyright © 2014 Microsoft Corporation.

```
  <StateValue>
    <BYTES>00 </BYTES>
  </StateValue>
</SessionStateData>
<SessionStateData>
  <StateId>
    <BYTE>05 </BYTE>
  </StateId>
  <StateLen>
    <BYTE>04 </BYTE>
  </StateLen>
  <StateValue>
    <BYTES>FF FF FF FF </BYTES>
  </StateValue>
</SessionStateData>
<SessionStateData>
  <StateId>
    <BYTE>06 </BYTE>
  </StateId>
  <StateLen>
    <BYTE>01 </BYTE>
  </StateLen>
  <StateValue>
    <BYTES>00 </BYTES>
  </StateValue>
</SessionStateData>
<SessionStateData>
  <StateId>
    <BYTE>07 </BYTE>
  </StateId>
  <StateLen>
    <BYTE>01 </BYTE>
  </StateLen>
  <StateValue>
    <BYTES>02 </BYTES>
  </StateValue>
</SessionStateData>
<SessionStateData>
  <StateId>
    <BYTE>08 </BYTE>
  </StateId>
  <StateLen>
    <BYTE>08 </BYTE>
  </StateLen>
  <StateValue>
    <BYTES>00 00 00 00 00 00 00 00 </BYTES>
  </StateValue>
</SessionStateData>
<SessionStateData>
  <StateId>
    <BYTE>09 </BYTE>
  </StateId>
  <StateLen>
    <BYTE>04 </BYTE>
  </StateLen>
  <StateValue>
    <BYTES>FF FF FF FF </BYTES>
  </StateValue>
</SessionStateData>
```


Copyright © 2014 Microsoft Corporation.

```
            </SessionStateDataSet>
          </InitSessionRecoveryData>
          <SessionRecoveryDataToBe>
            <Length>
              <DWORD>09 00 00 00 </DWORD>
            </Length>
            <RecoveryDatabase>
              <B_VARCHAR>
                <BYTE>00 </BYTE>
                <BYTES ascii="">
                </BYTES>
              </B_VARCHAR>
            </RecoveryDatabase>
            <RecoveryCollation>
              <BYTELEN>
                <BYTE>00 </BYTE>
              </BYTELEN>
              <BYTES>
              </BYTES>
            </RecoveryCollation>
            <RecoveryLanguage>
              <B_VARCHAR>
                <BYTE>00 </BYTE>
                <BYTES ascii="">
                </BYTES>
              </B_VARCHAR>
            </RecoveryLanguage>
            <SessionStateDataSet>
              <SessionStateData>
                <StateId>
                  <BYTE>09 </BYTE>
                </StateId>
                <StateLen>
                  <BYTE>04 </BYTE>
                </StateLen>
                <StateValue>
                  <BYTES>28 23 00 00 </BYTES>
                </StateValue>
              </SessionStateData>
            </SessionStateDataSet>
          </SessionRecoveryDataToBe>
        </FeatureData>
      </FeatureOpt>
      <FeatureOpt>
        <TERMINATOR>
          <BYTE>FF </BYTE>
        </TERMINATOR>
      </FeatureOpt>
    </FeatureExt>
  </Login7>
 </BufferData>
</tds>
```

4.17 FeatureExtAck with SessionRecovery Feature Data

A login response message that contains FeatureExt data for the SessionRecovery feature:

Copyright © 2014 Microsoft Corporation.

```
04 01 01 96 00 34 01 00 E3 1B 00 01 06 6D 00 61
00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
74 00 65 00 72 00 AB 58 00 45 16 00 00 02 00 25
00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
00 74 00 65 00 72 00 27 00 2E 00 00 00 00 00 00
00 E3 08 00 07 05 09 04 D0 00 34 00 E3 17 00 02
0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69
00 73 00 68 00 00 AB 5C 00 47 16 00 00 01 00 27
00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
00 6C 00 61 00 6E 00 67 00 75 00 61 00 67 00 65
00 20 00 73 00 65 00 74 00 74 00 69 00 6E 00 67
00 20 00 74 00 6F 00 20 00 75 00 73 00 5F 00 65
00 6E 00 67 00 6C 00 69 00 73 00 68 00 2E 00 00
00 00 00 00 00 AD 36 00 01 74 00 00 04 16 4D 00
69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00
20 00 53 00 51 00 4C 00 20 00 53 00 65 00 72 00
76 00 65 00 72 00 00 00 00 00 0B 00 08 C3 E3 13
00 04 04 34 00 30 00 39 00 36 00 04 34 00 30 00
39 00 36 00 AE 01 2E 00 00 00 00 09 00 60 81 14
FF E7 FF FF 00 02 02 07 01 04 01 00 05 04 FF FF
FF FF 06 01 00 07 01 02 08 08 00 00 00 00 00 00
00 00 09 04 28 23 00 00 FF FD 00 00 00 00 00 00
00 00 00 00 00 00
```

```
<tds version="latest">
  <BufferHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>01 </BYTE>
      <BYTE>96 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </BufferHeader>
  <BufferData>
    <TableResponse>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
```


Copyright © 2014 Microsoft Corporation.

```
          <USHORT>1B 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>01 06 6D 00 61 00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00 74 00 65 00
72 00 </BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <INFO>
        <TokenType>
          <BYTE>AB </BYTE>
        </TokenType>
        <Length>
          <USHORT>58 00 </USHORT>
        </Length>
        <Number>
          <LONG>45 16 00 00 </LONG>
        </Number>
        <State>
          <BYTE>02 </BYTE>
        </State>
        <Class>
          <BYTE>00 </BYTE>
        </Class>
        <MsgText>
          <US_UNICODE>
            <USHORTLEN>
              <USHORT>25 00 </USHORT>
            </USHORTLEN>
            <BYTES ascii="C.h.a.n.g.e.d. .d.a.t.a.b.a.s.e. .c.o.n.t.e.x.t. .t.o.
.'.m.a.s.t.e.r.'...">43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00 64 00 61 00 74 00 61 00
62 00 61 00 73 00 65 00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74 00 20 00 74 00 6F 00 20
00 27 00 6D 00 61 00 73 00 74 00 65 00 72 00 27 00 2E 00 </BYTES>
          </US_UNICODE>
        </MsgText>
        <ServerName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ServerName>
        <ProcName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ProcName>
        <LineNumber>
          <LONG>00 00 00 00 </LONG>
        </LineNumber>
      </INFO>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
```


Copyright © 2014 Microsoft Corporation.

```
        </TokenType>
        <Length>
          <USHORT>08 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>17 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>02 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00
</BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <INFO>
        <TokenType>
          <BYTE>AB </BYTE>
        </TokenType>
        <Length>
          <USHORT>5C 00 </USHORT>
        </Length>
        <Number>
          <LONG>47 16 00 00 </LONG>
        </Number>
        <State>
          <BYTE>01 </BYTE>
        </State>
        <Class>
          <BYTE>00 </BYTE>
        </Class>
        <MsgText>
          <US_UNICODE>
            <USHORTLEN>
              <USHORT>27 00 </USHORT>
            </USHORTLEN>
            <BYTES ascii="C.h.a.n.g.e.d. .l.a.n.g.u.a.g.e. .s.e.t.t.i.n.g. .t.o.
.u.s._.e.n.g.l.i.s.h...">43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00 6C 00 61 00 6E 00 67
00 75 00 61 00 67 00 65 00 20 00 73 00 65 00 74 00 74 00 69 00 6E 00 67 00 20 00 74 00 6F 00
20 00 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 2E 00 </BYTES>
          </US_UNICODE>
        </MsgText>
        <ServerName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ServerName>
        <ProcName>
          <B_UNICODE>
            <BYTELEN>
```


Copyright © 2014 Microsoft Corporation.

```
              <BYTE>00 </BYTE>
            </BYTELEN>
            <BYTES ascii="">
            </BYTES>
          </B_UNICODE>
        </ProcName>
        <LineNumber>
          <LONG>00 00 00 00 </LONG>
        </LineNumber>
      </INFO>
      <LOGINACK>
        <TokenType>
          <BYTE>AD </BYTE>
        </TokenType>
        <Length>
          <USHORT>36 00 </USHORT>
        </Length>
        <Interface>
          <BYTE>01 </BYTE>
        </Interface>
        <TDSVersion>
          <DWORD>FILTERED LATEST VERSION</DWORD>
        </TDSVersion>
        <ProgName>
          <B_UNICODE>
            <BYTELEN>
              <BYTE>16 </BYTE>
            </BYTELEN>
            <BYTES ascii="M.i.c.r.o.s.o.f.t. .S.Q.L. .S.e.r.v.e.r.....">4D 00 69 00 63 00 72
00 6F 00 73 00 6F 00 66 00 74 00 20 00 53 00 51 00 4C 00 20 00 53 00 65 00 72 00 76 00 65 00
72 00 00 00 00 00 </BYTES>
          </B_UNICODE>
        </ProgName>
        <ProgVersion>
          <DWORD>00 00 00 00 </DWORD>
        </ProgVersion>
      </LOGINACK>
      <ENVCHANGE>
        <TokenType>
          <BYTE>E3 </BYTE>
        </TokenType>
        <Length>
          <USHORT>13 00 </USHORT>
        </Length>
        <EnvChangeData>
          <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00 </BYTES>
        </EnvChangeData>
      </ENVCHANGE>
      <FEATUREEXTACK>
        <TokenType>
          <BYTE>AE </BYTE>
        </TokenType>
        <FeatureAckOpt>
          <FeatureId>
            <BYTE>01 </BYTE>
          </FeatureId>
          <FeatureAckDataLen>
            <DWORD>2E 00 00 00 </DWORD>
          </FeatureAckDataLen>
```


Copyright © 2014 Microsoft Corporation.

```
<SessionStateDataSet>
  <SessionStateData>
    <StateId>
      <BYTE>00 </BYTE>
    </StateId>
    <StateLen>
      <BYTE>09 </BYTE>
    </StateLen>
    <StateValue>
      <BYTES>00 60 81 14 FF E7 FF FF 00 </BYTES>
    </StateValue>
  </SessionStateData>
  <SessionStateData>
    <StateId>
      <BYTE>02 </BYTE>
    </StateId>
    <StateLen>
      <BYTE>02 </BYTE>
    </StateLen>
    <StateValue>
      <BYTES>07 01 </BYTES>
    </StateValue>
  </SessionStateData>
  <SessionStateData>
    <StateId>
      <BYTE>04 </BYTE>
    </StateId>
    <StateLen>
      <BYTE>01 </BYTE>
    </StateLen>
    <StateValue>
      <BYTES>00 </BYTES>
    </StateValue>
  </SessionStateData>
  <SessionStateData>
    <StateId>
      <BYTE>05 </BYTE>
    </StateId>
    <StateLen>
      <BYTE>04 </BYTE>
    </StateLen>
    <StateValue>
      <BYTES>FF FF FF FF </BYTES>
    </StateValue>
  </SessionStateData>
  <SessionStateData>
    <StateId>
      <BYTE>06 </BYTE>
    </StateId>
    <StateLen>
      <BYTE>01 </BYTE>
    </StateLen>
    <StateValue>
      <BYTES>00 </BYTES>
    </StateValue>
  </SessionStateData>
  <SessionStateData>
    <StateId>
      <BYTE>07 </BYTE>
```


Copyright © 2014 Microsoft Corporation.

```
            </StateId>
            <StateLen>
              <BYTE>01 </BYTE>
            </StateLen>
            <StateValue>
              <BYTES>02 </BYTES>
            </StateValue>
          </SessionStateData>
          <SessionStateData>
            <StateId>
              <BYTE>08 </BYTE>
            </StateId>
            <StateLen>
              <BYTE>08 </BYTE>
            </StateLen>
            <StateValue>
              <BYTES>00 00 00 00 00 00 00 00 </BYTES>
            </StateValue>
          </SessionStateData>
          <SessionStateData>
            <StateId>
              <BYTE>09 </BYTE>
            </StateId>
            <StateLen>
              <BYTE>04 </BYTE>
            </StateLen>
            <StateValue>
              <BYTES>28 23 00 00 </BYTES>
            </StateValue>
          </SessionStateData>
        </SessionStateDataSet>
      </FeatureAckOpt>
      <FeatureAckOpt>
        <TERMINATOR>
          <BYTE>FF </BYTE>
        </TERMINATOR>
      </FeatureAckOpt>
    </FEATUREEXTACK>
    <DONE>
      <TokenType>
        <BYTE>FD </BYTE>
      </TokenType>
      <Status>
        <USHORT>00 00 </USHORT>
      </Status>
      <CurCmd>
        <USHORT>00 00 </USHORT>
      </CurCmd>
      <DoneRowCount>
        <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
      </DoneRowCount>
    </DONE>
  </TableResponse>
 </BufferData>
</tds>
```


Copyright © 2014 Microsoft Corporation.

4.18 Table Response with SessionState Token Data

A response message that contains SessionState token data:

```
04 01 00 32 00 34 01 00 FD 01 00 BE 00 00 00 00
00 00 00 00 00 E4 0B 00 00 00 01 00 00 00 01 09
04 FF FF FF FF FD 00 00 FD 00 00 00 00 00 00 00
00 00
```

```
<tds version="latest">
  <BufferHeader>
    <Type>
      <BYTE>04 </BYTE>
    </Type>
    <Status>
      <BYTE>01 </BYTE>
    </Status>
    <Length>
      <BYTE>00 </BYTE>
      <BYTE>32 </BYTE>
    </Length>
    <SPID>
      <BYTE>00 </BYTE>
      <BYTE>00 </BYTE>
    </SPID>
    <Packet>
      <BYTE>01 </BYTE>
    </Packet>
    <Window>
      <BYTE>00 </BYTE>
    </Window>
  </BufferHeader>
  <BufferData>
    <TableResponse>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>01 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>BE 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
      <SESSIONSTATE>
        <TokenType>
          <BYTE>E4 </BYTE>
        </TokenType>
        <Length>
          <DWORD>0B 00 00 00 </DWORD>
        </Length>
        <SeqNo>
          <DWORD>01 00 00 00 </DWORD>
        </SeqNo>
```


Copyright © 2014 Microsoft Corporation.

```
        <Status>
          <BYTE>01 </BYTE>
        </Status>
        <SessionStateDataSet>
          <SessionStateData>
            <StateId>
              <BYTE>09 </BYTE>
            </StateId>
            <StateLen>
              <BYTE>04 </BYTE>
            </StateLen>
            <StateValue>
              <BYTES>FF FF FF FF </BYTES>
            </StateValue>
          </SessionStateData>
        </SessionStateDataSet>
      </SESSIONSTATE>
      <DONE>
        <TokenType>
          <BYTE>FD </BYTE>
        </TokenType>
        <Status>
          <USHORT>00 00 </USHORT>
        </Status>
        <CurCmd>
          <USHORT>FD 00 </USHORT>
        </CurCmd>
        <DoneRowCount>
          <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
        </DoneRowCount>
      </DONE>
    </TableResponse>
  </BufferData>
</tds>
```

4.19 Token Stream Communication

The following two examples highlight token stream communication. The packaging of these token streams into packets is not shown in this section. Actual TDS network data samples are available in section 4.

4.19.1 Sending a SQL Batch

In this example, a SQL statement is sent to the server and the results are sent to the client. The SQL statement is as follows:

```
SQLStatement =  select name, empid from employees
                update employees set salary = salary * 1.1
                select name from employees where department = 'HR'

Client:    SQLStatement

Server:    COLMETADATA   data stream
           ROW           data stream
           .
           .
```


Copyright © 2014 Microsoft Corporation.

```
ROW           data stream
DONE          data stream (with DONE_COUNT & DONE_MORE
                           bits set)
DONE          data stream (for UPDATE, with DONE_COUNT &
                           DONE_MORE bits set)
COLMETADATA   data stream
ROW           data stream
.
.
ROW           data stream
DONE          data stream (with DONE_COUNT bit set)
```

4.19.2 Out-of-Band Attention Signal

In this example, a SQL statement is sent to the server, yet before all the data has been returned an interrupt or "Attention Signal" is sent to the server. The client reads and discards any data received between the time the interrupt was sent and the interrupt acknowledgment was received. The interrupt acknowledgment from the server is a bit set in the status field of the DONE token.

```
Client:    select name, empid from employees

Server:    COLMETADATA   data stream
           ROW           data stream
           .
           .
           ROW           data stream

Client:    ATTENTION SENT
```

[The client reads and discards any data already buffered by the server until the acknowledgment is found. There might be or might not be a DONE token with the DONE_MORE bit clear prior to the DONE token with the DONE_ATTN bit set.]

```
Server:    DONE          data stream (with DONE_ATTN bit set)
```


Copyright © 2014 Microsoft Corporation.

5 Security

5.1 Security Considerations for Implementers

As previously described in this document, the TDS protocol provides facilities for authentication and channel encryption negotiation. If SSPI authentication is requested by the client application, then the exact choice of security mechanisms is determined by the SSPI layer. Likewise, although the decision as to whether channel encryption should be used is negotiated in the TDS layer, the exact choice of cipher suite is negotiated by the TLS/SSL layer. Likewise, although the decision as to whether federated authentication or SSPI authentication should be used may optionally be negotiated in the TDS layer, the exact choice of authentication mechanism is determined by either the SSPI layer or the federated authentication layer.

Copyright © 2014 Microsoft Corporation.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft SQL Server 7.0
- Microsoft SQL Server 2000
- Microsoft SQL Server 2005
- Microsoft SQL Server 2008
- Microsoft SQL Server 2008 R2
- Microsoft SQL Server 2012
- Microsoft SQL Server 2014

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.3: The following table outlines the SQL Server version and the corresponding TDS version.

SQL Server version	TDS version
SQL Server 7.0	7.0
SQL Server 2000	7.1
SQL Server 2000 SP1	7.1 Revision 1
SQL Server 2005	7.2
SQL Server 2008	7.3.A
SQL Server 2008 R2	7.3.B
SQL Server 2012	7.4
SQL Server 2014	7.4

<2> Section 2.1: Microsoft Windows Named Pipes in message mode [PIPE]. Please see [MSDN-NamedPipes] for additional information related to Microsoft-specific implementations.

<3> Section 2.1: VIA was deprecated in SQL Server 2012. This means that VIA will never be the underlying transport protocol if either the server or the client can support TDS versions up to TDS 7.4 or higher.

Copyright © 2014 Microsoft Corporation.

<4> Section 2.2.4.3: Not all pre-SQL Server 7.0 servers support the attention signal by using the message header. The older implementation was for the client to send a 1-byte message (no header) containing "A" by using the out-of-band write.

<5> Section 2.2.5.1.2: COLLATION represents a collation in SQL Server (described in [MSDN-Collation]). It can be either a SQL Server collation or a Windows collation.

Version can be of value 0, 1, or 2. A value of 0 denotes collations in SQL Server 2000. A value of 1 denotes collations in SQL Server 2005. A value of 2 denotes collations in SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, and SQL Server 2014.

The **GetLocaleInfo** Windows API can be used to retrieve information about the locale. In particular, querying for the LOCALE_IDEFAULTANSICODEPAGE locale information constant retrieves the code page information for the given locale.

For either collation type, the different comparison flags map to those defined as valid comparison flags for the **CompareString** Windows API.

However, for SQL collations with non-Unicode data, the SortId should be used to derive comparison information flags, such as whether, for a given SortId, a lowercase "a" equals an uppercase "A".

<6> Section 2.2.5.4.1: NULLTYPE can be sent to SQL Server (for example, in RPCRequest), but SQL Server never emits NULLTYPE data.

<7> Section 2.2.5.5.4: Microsoft implementations return an error if a client does send a raw collation within a sql_variant.

<8> Section 2.2.6.3: The version numbers used by clients are as follows.

SQL Server Version	Version Sent from Client to Server
SQL Server 7.0	0x00000070
SQL Server 2000	0x00000071
SQL Server 2000 SP1	0x01000071
SQL Server 2005	0x02000972
SQL Server 2008	0x03000A73
SQL Server 2008 R2	0x03000B73
SQL Server 2012 SQL Server 2014	0x04000074

<9> Section 2.2.6.4: The US_SUBBUILD returned by SQL Server is always 0.

<10> Section 2.2.6.4: In SQL Server 2012 and SQL Server 2014, the server always sends the value 0 for the INSTOPT option when the string specified in the client's INSTOPT option is "MSSQLServer". The reason for this is that "MSSQLServer" is the name of a default instance, and "MSSQLServer" may be provided by the client even in the absence of an explicit instance name. SQL Server 2000, SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2, which support the INSTOPT field always validate the client-specified string against the server's instance name.

<11> Section 2.2.7.5: The 0x4: DONE_INXACT bit is not set by SQL Server and should be considered reserved for future use.

Copyright © 2014 Microsoft Corporation.

<12> Section 2.2.7.5: The DONE token is usually sent after login has succeeded. In this case, the negotiated TDS version is known, and the client can determine whether DoneRowCount is LONG or ULONGLONG.

However, when login fails for any reason, SQL Server may also send an error message followed by a DONE token. In this case, the server should have already done TDS version negotiation and has to send DoneRowCount as LONG or ULONGLONG based on the negotiated TDS version.

However, sometimes the client cannot determine the server TDS version and cannot determine whether LONG or ULONGLONG is expected for DoneRowCount. If the client TDS level is 7.0 or 7.1, DoneRowCount is always LONG. If the client TDS level is 7.2, 7.3.A, 7.3.B, or 7.4, the DoneRowCount can be LONG or ULONGLONG depending on which version of the server the client is connecting to.

SNAC and SQLClient use the VERSION option in the prelogin response to detect whether DoneRowCount is LONG or ULONGLONG. It is ULONGLONG if VERSION in the prelogin response indicates that the server is SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, or SQL Server 2014. Otherwise, DoneRowCount is LONG.

A third-party implementation should have its own logic to detect whether DoneRowCount is LONG or ULONGLONG or to make the client able to handle both LONG and ULONGLONG. In any implementation, before the client performs this task, the server performs TDS version negotiation and determines whether to send LONG or ULONGLONG.

<13> Section 2.2.7.6: The 0x4: DONE_INXACT bit is not set by SQL Server and should be considered reserved for future use.

<14> Section 2.2.7.7: The 0x4: DONE_INXACT bit is not set by SQL Server and should be considered reserved for future use.

<15> Section 2.2.7.8: The **Type 16: Transaction Manager Address** is not used by SQL Server.

<16> Section 2.2.7.9: Numbers less than 20001 are reserved by SQL Server.

<17> Section 2.2.7.9: SQL Server does not raise system errors with severities of 0 through 9.

<18> Section 2.2.7.9: For compatibility reasons, SQL Server converts severity 10 to severity 0 before returning the error information to the calling application.

<19> Section 2.2.7.11: Numbers less than 20001 are reserved by SQL Server.

<20> Section 2.2.7.12: The following table shows the values in network transfer format.

SQL Server	Client to server	Server to client
SQL Server 7.0	0x00000070	0x07000000
SQL Server 2000	0x00000071	0x07010000
SQL Server 2000 SP1	0x01000071	0x71000001
SQL Server 2005	0x02000972	0x72090002
SQL Server 2008*	0x03000A73	0x730A0003
SQL Server 2008 R2	0x03000B73	0x730B0003
SQL Server 2012	0x04000074	0x74000004

Copyright © 2014 Microsoft Corporation.

SQL Server	Client to server	Server to client
SQL Server 2014		

*SQL Server 2008 TDS version 0x03000A73 does not include support for NBCROW and fSparseColumnSet.

<21> Section 3.2.2: In Microsoft implementations, the default value for the MDAC/WDAC and SNAC Client Request Timers is zero, which is interpreted as no timeout. For a SqlClient Client Request, the default value is 30 seconds. For a description of the data access drivers, see [MSDN-MDAC].

<22> Section 3.2.2: In Microsoft implementations, the default setting for MDAC/WDAC and SNAC Cancel Timer values is 120 seconds. For a SqlClient Cancel Timer, the default value is 5 seconds. For a description of the data access drivers, see [MSDN-MDAC].

Copyright © 2014 Microsoft Corporation.

7 Change Tracking

This section identifies changes that were made to the [MS-TDS] protocol document between the February 2014 and May 2014 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.

Copyright © 2014 Microsoft Corporation.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.1 Glossary	1861637 Specified relevant definition of the term "interface".	Y	Content updated.
2.2.3.1.1 Type	1861680 Changed "Request Completion" to "Response Completion ("DONE")".	Y	Content updated.
2.2.4 Packet Data Token and Tokenless Data Streams	1861680 Changed "Request Completion" to "Response Completion ("DONE")".	Y	Content updated.
2.2.6.3 LOGIN7	1861683 Changed "fFederatedAuthEcho" to "fFedAuthEcho".	Y	Content updated.
2.2.6.3 LOGIN7	806250 Clarified that both CURSOR_CLOSE_ON_COMMIT and IMPLICIT_TRANSACTIONS are set to OFF for OptionFlags2 fODBC and TypeFlags fOLEDB flags.	Y	Content updated.
2.2.6.7 SSPI Message	1861687 Changed "SSPIMessage" to "SSPI" or "SSPI message" as appropriate to context.	Y	Content updated.
6 Appendix A: Product Behavior	1861692 Added SQL Server 7.0 to the list of applicable products.	Y	Content updated.

Copyright © 2014 Microsoft Corporation.

8 Index

A

Abstract data model
  client (section 3.1.1 111, section 3.2.1 116)
  server (section 3.1.1 111, section 3.3.1 124)
ALL_HEADERS rule definition
  overview 32
  Query Notifications header 33
  Transaction Descriptor header 34
Applicability 15
Attention message 17
Attention request example 157
Attention signal - out-of-band 183
Attention tokens 26

C

Capability negotiation 15
Change tracking 189
Client
  abstract data model (section 3.1.1 111, section 3.2.1 116)
  higher-layer triggered events (section 3.1.4 111, section 3.2.4 117)
  initialization (section 3.1.3 111, section 3.2.3 117)
  local events (section 3.1.7 115, section 3.2.7 123)
  message processing 111
    Final state 122
    Logged In state 121
    overview 119
    Sent Attention state 122
    Sent Client Request state 122
    Sent Initial PRELOGIN Packet state 119
    Sent LOGIN7 Record with SPNEGO Packet state 121
    Sent LOGIN7 Record with Standard Login state 120
    Sent TLS/SSL Negotiation Packet state 120
  messages
    Attention 17
    login 17
    overview 16
    pre-login 17
    remote procedure call 17
    SQL command 17
    SQL command with binary data 17
    transaction manager request 18
  overview (section 3.1 111, section 3.2 115)
  sequencing rules 111
    Final state 122
    Logged In state 121
    overview 119
    Sent Attention state 122
    Sent Client Request state 122
    Sent Initial PRELOGIN Packet state 119
    Sent LOGIN7 Record with SPNEGO Packet state 121
    Sent LOGIN7 Record with Standard Login state 120
    Sent TLS/SSL Negotiation Packet state 120
  timer events (section 3.1.6 115, section 3.2.6 123)
  timers (section 3.1.2 111, section 3.2.2 117)
Client Request Execution state 129

D

Data model - abstract
  client (section 3.1.1 111, section 3.2.1 116)
  server (section 3.1.1 111, section 3.3.1 124)
Data stream types
  data type dependent data streams 31
  unknown-length data streams 30
  variable-length data streams 30
Data type definitions
  fixed-length data types 35
  overview 35
  partially length-prefixed data types 39
  SQL_VARIANT 42
  Table Valued Parameter
    metadata 43
    optional metadata tokens 46
    overview 43
    TDS type restrictions 48
  UDT Assembly Information 41
  variable-length data types 36
  XML data type 42
DONE tokens 26

E

Error messages 19
Examples
  attention request 157
  login request 131
  login request with federated authentication 134
  login response 141
  login response with federated authentication 146
  overview 130
  pre-login request 130
  RPC client request 153
  RPC server response 156
  SQL batch client request 150
  SQL batch server response 152
  SQL command with binary data 158
  SSPI message 157
  transaction manager request 160

F

Fields - vendor-extensible 15
Final state (section 3.2.5.9 122, section 3.3.5.8 129)

Copyright © 2014 Microsoft Corporation.

Fixed-length token 25

G

Glossary 8
Grammar definition - token description
  data buffer stream tokens 50
  data stream types
    data type dependent data streams 31
    unknown-length data streams 30
    variable-length data streams 30
  data type definitions
    fixed-length data types 35
    overview 35
    partially length-prefixed data types 39
    SQL_VARIANT 42
    Table Valued Parameter 43
    UDT Assembly Information 41
    variable-length data types 36
    XML data type 42
  general rules
    collation rule definition 29
    least significant bit order 29
    overview 26
  overview 26
  packet data stream headers
    overview 32
    Query Notifications header 33
    Transaction Descriptor header 34
  TYPE_INFO rule definition 49

H

Higher-layer triggered events
  client (section 3.1.4 111, section 3.2.4 117)
  server (section 3.1.4 111, section 3.3.4 125)

I

Implementer - security considerations 184
Informational messages 19
Informative references 11
Initial state 125
Initialization
  client (section 3.1.3 111, section 3.2.3 117)
  server (section 3.1.3 111, section 3.3.3 125)
Introduction 8

L

Local events
  client (section 3.1.7 115, section 3.2.7 123)
  server (section 3.1.7 115, section 3.3.7 129)
Logged In state (section 3.2.5.5 121, section 3.3.5.5 128)
Login Ready state 126
Login request example 131
Login request with federated authentication example 134
Login response example 141
Login response with federated authentication example 146

M

Message processing
  client 111
    Final state 122
    Logged In state 121
    overview 119
    Sent Attention state 122
    Sent Client Request state 122
    Sent Initial PRELOGIN Packet state 119
    Sent LOGIN7 Record with SPNEGO Packet state 121
    Sent LOGIN7 Record with Standard Login state 120
    Sent TLS/SSL Negotiation Packet state 120
  server 111
    Client Request Execution state 129
    Final state 129
    Initial state 125
    Logged In state 128
    Login Ready state 126
    overview 125
    SPNEGO Negotiation state 128
    TLS/SSL Negotiation state 126
Messages
  overview 16
  syntax
    client messages 16
    grammar definition for token description 26
    overview 16
    packet data token and tokenless data streams 23
    packet data token stream definition 76
    packet header message type - stream definition 51
    packets 20
    server messages 18
  transport 16

N

Normative references 10

O

Out-of-band attention signal 183
Overview (synopsis) 12

P

Packet data - token and tokenless streams 23
Packet data - token stream definition
  ALTMETADATA 76
  ALTROW 79
  COLINFO 80
  COLMETADATA 81
  DONE 83
  DONEINPROC 85
  DONEPROC 86
  ENVCHANGE 87
  ERROR 91
  FEATUREEXTACK 94

Copyright © 2014 Microsoft Corporation.

INFO 96
LOGINACK 97
NBCROW 98
OFFSET 100
ORDER 101
overview 76
RETURNSTATUS 102
RETURNVALUE 102
ROW 105
SESSIONSTATE 106
SSPI 107
Table Valued Parameter row 109
TABNAME 108
Packet data stream headers
overview 32
Query Notifications header 33
Transaction Descriptor header 34
Packet header message type - stream definition
BulkLoad - UpdateText/WriteText 51
BulkLoadBCP 51
FeatureExt with SessionRecovery feature data 169
FeatureExtAck with SessionRecovery feature data 174
LOGIN7 52
PRELOGIN 64
RPCRequest 68
SparseColumn select statement 163
SQLBatch 70
SSPI message 71
Table response with SessionState token data 181
transaction manager request 72
TVP insert statement 161
Packets
overview 20
packet data 23
packet header
Length 22
overview 20
PacketID 23
SPID 22
Status 22
Type 20
Window 23
Preconditions 14
Pre-login request example 130
Prerequisites 14
Product behavior 185

Q

Query Notifications header 33

R

References
informative 11
normative 10
Relationship to other protocols 14
Remote procedure call 17
RPC client request example 153
RPC server response example 156

S

Security - implementer considerations 184
Sending an SQL batch 182
Sent Attention state 122
Sent Client Request state 122
Sent Initial PRELOGIN Packet state 119
Sent LOGIN7 Record with SPNEGO Packet state 121
Sent LOGIN7 Record with Standard Login state 120
Sent TLS/SSL Negotiation Packet state 120
Sequencing rules
client 111
Final state 122
Logged In state 121
overview 119
Sent Attention state 122
Sent Client Request state 122
Sent Initial PRELOGIN Packet state 119
Sent LOGIN7 Record with SPNEGO Packet state 121
Sent LOGIN7 Record with Standard Login state 120
Sent TLS/SSL Negotiation Packet state 120
server 111
Client Request Execution state 129
Final state 129
Initial state 125
Logged In state 128
Login Ready state 126
overview 125
SPNEGO Negotiation state 128
TLS/SSL Negotiation state 126
Server
abstract data model (section 3.1.1 111, section 3.3.1 124)
higher-layer triggered events (section 3.1.4 111, section 3.3.4 125)
initialization (section 3.1.3 111, section 3.3.3 125)
local events (section 3.1.7 115, section 3.3.7 129)
message processing 111
Client Request Execution state 129
Final state 129
Initial state 125
Logged In state 128
Login Ready state 126
overview 125
SPNEGO Negotiation state 128
TLS/SSL Negotiation state 126
messages
attention acknowledgment 20
error and informational messages 19
login response 18
overview 18
pre-login response 18
response completion ("DONE") 19
return parameters 19
return status 19
row data 19
overview (section 3.1 111, section 3.3 123)

Copyright © 2014 Microsoft Corporation.

sequencing rules 111
Client Request Execution state 129
Final state 129
Initial state 125
Logged In state 128
Login Ready state 126
overview 125
SPNEGO Negotiation state 128
TLS/SSL Negotiation state 126
timer events (section 3.1.6 115, section 3.3.6 129)
timers (section 3.1.2 111, section 3.3.2 125)
SPNEGO Negotiation state 128
SQL batch - sending 182
SQL batch client request example 150
SQL batch server response example 152
SQL command 17
SQL command with binary data 17
SQL command with binary data example 158
SSPI message example 157
Standards assignments 15
Syntax
client messages
Attention 17
login 17
overview 16
pre-login 17
remote procedure call 17
SQL command 17
SQL command with binary data 17
transaction manager request 18
grammar definition for token description
data buffer stream tokens 50
data stream types 30
data type definitions 35
general rules 26
overview 26
packet data stream headers 32
TYPE_INFO rule definition 49
overview 16
packet data token and tokenless data streams
DONE and attention tokens 26
overview 23
token stream 24
token stream examples 182
tokenless stream 24
packet data token stream definition
ALTMETADATA 76
ALTROW 79
COLINFO 80
COLMETADATA 81
DONE 83
DONEINPROC 85
DONEPROC 86
ENVCHANGE 87
ERROR 91
FEATUREEXTACK 94
INFO 96
LOGINACK 97
NBCROW 98
OFFSET 100
ORDER 101
overview 76
RETURNSTATUS 102
RETURNVALUE 102
ROW 105
SESSIONSTATE 106
SSPI 107
Table Valued Parameter row 109
TABNAME 108
packet header message type - stream definition
BulkLoad - UpdateText/WriteText 51
BulkLoadBCP 51
FeatureExt with SessionRecovery feature data 169
FeatureExtAck with SessionRecovery feature data 174
LOGIN7 52
PRELOGIN 64
RPCRequest 68
SparseColumn select statement 163
SQLBatch 70
SSPI message 71
Table response with SessionState token data 181
transaction manager request 72
TVP insert statement 161
packets
overview 20
packet data 23
packet header 20
server messages
attention acknowledgment 20
error and informational messages 19
login response 18
overview 18
pre-login response 18
response completion ("DONE") 19
return parameters 19
return status 19
row data 19

T

Timer events
client (section 3.1.6 115, section 3.2.6 123)
server (section 3.1.6 115, section 3.3.6 129)
Timers
client (section 3.1.2 111, section 3.2.2 117)
server (section 3.1.2 111, section 3.3.2 125)
TLS/SSL Negotiation state 126
Token data stream
overview 24
token definition
fixed-length token 25
overview 24
variable-count tokens 26
variable-length tokens 25
zero-length token 24
Token data stream definition
ALTMETADATA 76
ALTROW 79
COLINFO 80

Copyright © 2014 Microsoft Corporation.

COLMETADATA 81
DONE 83
DONEINPROC 85
DONEPROC 86
ENVCHANGE 87
ERROR 91
FEATUREEXTACK 94
INFO 96
LOGINACK 97
NBCROW 98
OFFSET 100
ORDER 101
overview 76
RETURNSTATUS 102
RETURNVALUE 102
ROW 105
SESSIONSTATE 106
SSPI 107
Table Valued Parameter row 109
TABNAME 108
Token data stream examples
out-of-band attention signal 183
overview 182
sending an SQL batch 182
Token description - grammar definition
data buffer stream tokens 50
data stream types
data type dependent data streams 31
unknown-length data streams 30
variable-length data streams 30
data type definitions
fixed-length data types 35
overview 35
partially length-prefixed data types 39
SQL_VARIANT 42
Table Valued Parameter 43
UDT Assembly Information 41
variable-length data types 36
XML data type 42
general rules
collation rule definition 29
least significant bit order 29
overview 26
overview 26
packet data stream headers
overview 32
Query Notifications header 33
Transaction Descriptor header 34
TYPE_INFO rule definition 49
Tokenless data stream 24
Tracking changes 189
Transaction Descriptor header 34
Transaction manager request 18
Transaction manager request example 160
Transport 16
Triggered events - higher-layer
client (section 3.1.4 111, section 3.2.4 117)
server (section 3.1.4 111, section 3.3.4 125)

U

Unknown-length data streams 30

V

Variable-count tokens 26
Variable-length data streams 30
Variable-length tokens 25
Vendor-extensible fields 15
Versioning 15

Z

Zero-length token 24

Copyright © 2014 Microsoft Corporation.