**[MS-TDS]:**

**Tabular Data Stream Protocol**

Intellectual Property Rights Notice for Open Specifications Documentation

* **Technical Documentation.** Microsoft publishes Open Specifications documentation (“this documentation”) for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
* **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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
* **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
* **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft [Open Specifications Promise](https://go.microsoft.com/fwlink/?LinkId=214445) or the [Microsoft Community Promise](https://go.microsoft.com/fwlink/?LinkId=214448). If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting [iplg@microsoft.com](mailto:iplg@microsoft.com).
* **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the [Patent Map](https://msdn.microsoft.com/en-us/openspecifications/dn750984).
* **Trademarks**. The names of companies and products contained in this documentation might 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](https://www.microsoft.com/trademarks).
* **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are 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 as specifically described above, whether by implication, estoppel, or otherwise.

**Tools**. The Open Specifications documentation does 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 documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

**Support.** For questions and support, please contact [dochelp@microsoft.com](mailto:dochelp@microsoft.com).

**Revision Summary**

| Date | Revision History | Revision Class | Comments |
| --- | --- | --- | --- |
| 3/14/2008 | 0.1 | Major | Initial Availability. |
| 6/20/2008 | 0.1.1 | Editorial | Changed language and formatting in the technical content. |
| 7/25/2008 | 0.1.2 | Editorial | Changed language and formatting in the technical content. |
| 8/29/2008 | 0.1.3 | Editorial | Changed language and formatting in the technical content. |
| 10/24/2008 | 0.1.4 | Editorial | Changed language and formatting in the technical content. |
| 12/5/2008 | 0.2 | Minor | Clarified the meaning of the technical content. |
| 1/16/2009 | 0.3 | Minor | Clarified the meaning of the technical content. |
| 2/27/2009 | 0.4 | Minor | Clarified the meaning of the technical content. |
| 4/10/2009 | 0.5 | Minor | Clarified the meaning of the technical content. |
| 5/22/2009 | 0.5.1 | Editorial | Changed language and formatting in the technical content. |
| 7/2/2009 | 1.0 | Major | Updated and revised the technical content. |
| 8/14/2009 | 1.1 | Minor | Clarified the meaning of the technical content. |
| 9/25/2009 | 2.0 | Major | Updated and revised the technical content. |
| 11/6/2009 | 3.0 | Major | Updated and revised the technical content. |
| 12/18/2009 | 4.0 | Major | Updated and revised the technical content. |
| 1/29/2010 | 4.1 | Minor | Clarified the meaning of the technical content. |
| 3/12/2010 | 5.0 | Major | Updated and revised the technical content. |
| 4/23/2010 | 6.0 | Major | Updated and revised the technical content. |
| 6/4/2010 | 7.0 | Major | Updated and revised the technical content. |
| 7/16/2010 | 8.0 | Major | Updated and revised the technical content. |
| 8/27/2010 | 8.0 | None | No changes to the meaning, language, or formatting of the technical content. |
| 10/8/2010 | 9.0 | Major | Updated and revised the technical content. |
| 11/19/2010 | 9.0 | None | No changes to the meaning, language, or formatting of the technical content. |
| 1/7/2011 | 9.1 | Minor | Clarified the meaning of the technical content. |
| 2/11/2011 | 9.2 | Minor | Clarified the meaning of the technical content. |
| 3/25/2011 | 9.3 | Minor | Clarified the meaning of the technical content. |
| 5/6/2011 | 9.4 | Minor | Clarified the meaning of the technical content. |
| 6/17/2011 | 10.0 | Major | Updated and revised the technical content. |
| 9/23/2011 | 11.0 | Major | Updated and revised the technical content. |
| 12/16/2011 | 12.0 | Major | Updated and revised the technical content. |
| 3/30/2012 | 12.1 | Minor | Clarified the meaning of the technical content. |
| 7/12/2012 | 12.2 | Minor | Clarified the meaning of the technical content. |
| 10/25/2012 | 12.2 | None | No changes to the meaning, language, or formatting of the technical content. |
| 1/31/2013 | 13.0 | Major | Updated and revised the technical content. |
| 8/8/2013 | 14.0 | Major | Updated and revised the technical content. |
| 11/14/2013 | 15.0 | Major | Updated and revised the technical content. |
| 2/13/2014 | 16.0 | Major | Updated and revised the technical content. |
| 5/15/2014 | 17.0 | Major | Updated and revised the technical content. |
| 6/30/2015 | 18.0 | Major | Significantly changed the technical content. |
| 10/16/2015 | 19.0 | Major | Significantly changed the technical content. |
| 5/10/2016 | 19.0 | None | No changes to the meaning, language, or formatting of the technical content. |
| 7/14/2016 | 19.0 | None | No changes to the meaning, language, or formatting of the technical content. |
| 3/16/2017 | 20.0 | Major | Significantly changed the technical content. |
| 6/1/2017 | 20.0 | None | No changes to the meaning, language, or formatting of the technical content. |
| 8/16/2017 | 21.0 | Major | Significantly changed the technical content. |
| 9/15/2017 | 22.0 | Major | Significantly changed the technical content. |
| 12/1/2017 | 23.0 | Major | Significantly changed the technical content. |
| 3/16/2018 | 24.0 | Major | Significantly changed the technical content. |
| 9/12/2018 | 25.0 | Major | Significantly changed the technical content. |
| 3/13/2019 | 26.0 | Major | Significantly changed the technical content. |

Table of Contents

[1 Introduction 8](#_Toc2248712)

[1.1 Glossary 8](#_Toc2248713)

[1.2 References 10](#_Toc2248714)

[1.2.1 Normative References 10](#_Toc2248715)

[1.2.2 Informative References 11](#_Toc2248716)

[1.3 Overview 13](#_Toc2248717)

[1.4 Relationship to Other Protocols 14](#_Toc2248718)

[1.5 Prerequisites/Preconditions 14](#_Toc2248719)

[1.6 Applicability Statement 15](#_Toc2248720)

[1.7 Versioning and Capability Negotiation 15](#_Toc2248721)

[1.8 Vendor-Extensible Fields 16](#_Toc2248722)

[1.9 Standards Assignments 16](#_Toc2248723)

[2 Messages 17](#_Toc2248724)

[2.1 Transport 17](#_Toc2248725)

[2.2 Message Syntax 17](#_Toc2248726)

[2.2.1 Client Messages 17](#_Toc2248727)

[2.2.1.1 Pre-Login 18](#_Toc2248728)

[2.2.1.2 Login 18](#_Toc2248729)

[2.2.1.3 Federated Authentication Token 18](#_Toc2248730)

[2.2.1.4 SQL Batch 18](#_Toc2248731)

[2.2.1.5 Bulk Load 18](#_Toc2248732)

[2.2.1.6 Remote Procedure Call 19](#_Toc2248733)

[2.2.1.7 Attention 19](#_Toc2248734)

[2.2.1.8 Transaction Manager Request 19](#_Toc2248735)

[2.2.2 Server Messages 19](#_Toc2248736)

[2.2.2.1 Pre-Login Response 20](#_Toc2248737)

[2.2.2.2 Login Response 20](#_Toc2248738)

[2.2.2.3 Federated Authentication Information 20](#_Toc2248739)

[2.2.2.4 Row Data 20](#_Toc2248740)

[2.2.2.5 Return Status 20](#_Toc2248741)

[2.2.2.6 Return Parameters 20](#_Toc2248742)

[2.2.2.7 Response Completion 21](#_Toc2248743)

[2.2.2.8 Error and Info 21](#_Toc2248744)

[2.2.2.9 Attention Acknowledgment 21](#_Toc2248745)

[2.2.3 Packets 21](#_Toc2248746)

[2.2.3.1 Packet Header 22](#_Toc2248747)

[2.2.3.1.1 Type 22](#_Toc2248748)

[2.2.3.1.2 Status 23](#_Toc2248749)

[2.2.3.1.3 Length 24](#_Toc2248750)

[2.2.3.1.4 SPID 24](#_Toc2248751)

[2.2.3.1.5 PacketID 24](#_Toc2248752)

[2.2.3.1.6 Window 24](#_Toc2248753)

[2.2.3.2 Packet Data 24](#_Toc2248754)

[2.2.4 Packet Data Token and Tokenless Data Streams 25](#_Toc2248755)

[2.2.4.1 Tokenless Stream 25](#_Toc2248756)

[2.2.4.2 Token Stream 26](#_Toc2248757)

[2.2.4.2.1 Token Definition 26](#_Toc2248758)

[2.2.4.2.1.1 Zero Length Token(xx01xxxx) 26](#_Toc2248759)

[2.2.4.2.1.2 Fixed Length Token(xx11xxxx) 26](#_Toc2248760)

[2.2.4.2.1.3 Variable Length Tokens(xx10xxxx) 27](#_Toc2248761)

[2.2.4.2.1.4 Variable Count Tokens(xx00xxxx) 27](#_Toc2248762)

[2.2.4.3 Done and Attention Tokens 27](#_Toc2248763)

[2.2.5 Grammar Definition for Token Description 27](#_Toc2248764)

[2.2.5.1 General Rules 27](#_Toc2248765)

[2.2.5.1.1 Least Significant Bit Order 30](#_Toc2248766)

[2.2.5.1.2 Collation Rule Definition 30](#_Toc2248767)

[2.2.5.2 Data Stream Types 31](#_Toc2248768)

[2.2.5.2.1 Unknown Length Data Streams 31](#_Toc2248769)

[2.2.5.2.2 Variable-Length Data Streams 31](#_Toc2248770)

[2.2.5.2.3 Data Type Dependent Data Streams 31](#_Toc2248771)

[2.2.5.3 Packet Data Stream Headers - ALL\_HEADERS Rule Definition 33](#_Toc2248772)

[2.2.5.3.1 Query Notifications Header 34](#_Toc2248773)

[2.2.5.3.2 Transaction Descriptor Header 34](#_Toc2248774)

[2.2.5.3.3 Trace Activity Header 35](#_Toc2248775)

[2.2.5.4 Data Type Definitions 35](#_Toc2248776)

[2.2.5.4.1 Fixed-Length Data Types 35](#_Toc2248777)

[2.2.5.4.2 Variable-Length Data Types 36](#_Toc2248778)

[2.2.5.4.3 Partially Length-Prefixed Data Types 39](#_Toc2248779)

[2.2.5.5 Data Type Details 39](#_Toc2248780)

[2.2.5.5.1 System Data Type Values 39](#_Toc2248781)

[2.2.5.5.1.1 Integers 39](#_Toc2248782)

[2.2.5.5.1.2 Timestamp 39](#_Toc2248783)

[2.2.5.5.1.3 Character and Binary Strings 40](#_Toc2248784)

[2.2.5.5.1.4 Fixed-Point Numbers 40](#_Toc2248785)

[2.2.5.5.1.5 Floating-Point Numbers 40](#_Toc2248786)

[2.2.5.5.1.6 Decimal/Numeric 40](#_Toc2248787)

[2.2.5.5.1.7 GUID 40](#_Toc2248788)

[2.2.5.5.1.8 Date/Times 40](#_Toc2248789)

[2.2.5.5.2 Common Language Runtime (CLR) Instances 41](#_Toc2248790)

[2.2.5.5.3 XML Values 42](#_Toc2248791)

[2.2.5.5.4 SQL\_VARIANT Values 42](#_Toc2248792)

[2.2.5.5.5 Table Valued Parameter (TVP) Values 43](#_Toc2248793)

[2.2.5.5.5.1 Metadata 43](#_Toc2248794)

[2.2.5.5.5.2 Optional Metadata Tokens 45](#_Toc2248795)

[2.2.5.5.5.3 TDS Type Restrictions 47](#_Toc2248796)

[2.2.5.6 Type Info Rule Definition 48](#_Toc2248797)

[2.2.5.7 Encryption Key Rule Definition 49](#_Toc2248798)

[2.2.5.8 Data Packet Stream Tokens 50](#_Toc2248799)

[2.2.6 Packet Header Message Type Stream Definition 50](#_Toc2248800)

[2.2.6.1 Bulk Load BCP 50](#_Toc2248801)

[2.2.6.2 Bulk Load Update Text/Write Text 51](#_Toc2248802)

[2.2.6.3 Federated Authentication Token 52](#_Toc2248803)

[2.2.6.4 LOGIN7 53](#_Toc2248804)

[2.2.6.5 PRELOGIN 65](#_Toc2248805)

[2.2.6.6 RPC Request 69](#_Toc2248806)

[2.2.6.7 SQLBatch 73](#_Toc2248807)

[2.2.6.8 SSPI Message 73](#_Toc2248808)

[2.2.6.9 Transaction Manager Request 74](#_Toc2248809)

[2.2.7 Packet Data Token Stream Definition 77](#_Toc2248810)

[2.2.7.1 ALTMETADATA 78](#_Toc2248811)

[2.2.7.2 ALTROW 80](#_Toc2248812)

[2.2.7.3 COLINFO 81](#_Toc2248813)

[2.2.7.4 COLMETADATA 82](#_Toc2248814)

[2.2.7.5 DONE 85](#_Toc2248815)

[2.2.7.6 DONEINPROC 87](#_Toc2248816)

[2.2.7.7 DONEPROC 88](#_Toc2248817)

[2.2.7.8 ENVCHANGE 89](#_Toc2248818)

[2.2.7.9 ERROR 93](#_Toc2248819)

[2.2.7.10 FEATUREEXTACK 95](#_Toc2248820)

[2.2.7.11 FEDAUTHINFO 98](#_Toc2248821)

[2.2.7.12 INFO 100](#_Toc2248822)

[2.2.7.13 LOGINACK 101](#_Toc2248823)

[2.2.7.14 NBCROW 102](#_Toc2248824)

[2.2.7.15 OFFSET 104](#_Toc2248825)

[2.2.7.16 ORDER 104](#_Toc2248826)

[2.2.7.17 RETURNSTATUS 105](#_Toc2248827)

[2.2.7.18 RETURNVALUE 106](#_Toc2248828)

[2.2.7.19 ROW 108](#_Toc2248829)

[2.2.7.20 SESSIONSTATE 109](#_Toc2248830)

[2.2.7.21 SSPI 111](#_Toc2248831)

[2.2.7.22 TABNAME 111](#_Toc2248832)

[2.2.7.23 TVP\_ROW 112](#_Toc2248833)

[3 Protocol Details 114](#_Toc2248834)

[3.1 Common Details 114](#_Toc2248835)

[3.1.1 Abstract Data Model 114](#_Toc2248836)

[3.1.2 Timers 114](#_Toc2248837)

[3.1.3 Initialization 114](#_Toc2248838)

[3.1.4 Higher-Layer Triggered Events 114](#_Toc2248839)

[3.1.5 Message Processing Events and Sequencing Rules 114](#_Toc2248840)

[3.1.6 Timer Events 119](#_Toc2248841)

[3.1.7 Other Local Events 119](#_Toc2248842)

[3.2 Client Details 120](#_Toc2248843)

[3.2.1 Abstract Data Model 120](#_Toc2248844)

[3.2.2 Timers 121](#_Toc2248845)

[3.2.3 Initialization 121](#_Toc2248846)

[3.2.4 Higher-Layer Triggered Events 122](#_Toc2248847)

[3.2.5 Message Processing Events and Sequencing Rules 123](#_Toc2248848)

[3.2.5.1 Sent Initial PRELOGIN Packet State 123](#_Toc2248849)

[3.2.5.2 Sent TLS/SSL Negotiation Packet State 124](#_Toc2248850)

[3.2.5.3 Sent LOGIN7 Record with Complete Authentication Token State 125](#_Toc2248851)

[3.2.5.4 Sent LOGIN7 Record with SPNEGO Packet State 125](#_Toc2248852)

[3.2.5.5 Sent LOGIN7 Record with Federated Authentication Information Request State 126](#_Toc2248853)

[3.2.5.6 Logged In State 126](#_Toc2248854)

[3.2.5.7 Sent Client Request State 126](#_Toc2248855)

[3.2.5.8 Sent Attention State 126](#_Toc2248856)

[3.2.5.9 Routing Completed State 127](#_Toc2248857)

[3.2.5.10 Final State 127](#_Toc2248858)

[3.2.6 Timer Events 127](#_Toc2248859)

[3.2.7 Other Local Events 127](#_Toc2248860)

[3.3 Server Details 127](#_Toc2248861)

[3.3.1 Abstract Data Model 128](#_Toc2248862)

[3.3.2 Timers 129](#_Toc2248863)

[3.3.3 Initialization 129](#_Toc2248864)

[3.3.4 Higher-Layer Triggered Events 129](#_Toc2248865)

[3.3.5 Message Processing Events and Sequencing Rules 129](#_Toc2248866)

[3.3.5.1 Initial State 129](#_Toc2248867)

[3.3.5.2 TLS/SSL Negotiation State 130](#_Toc2248868)

[3.3.5.3 Login Ready State 130](#_Toc2248869)

[3.3.5.4 SPNEGO Negotiation State 132](#_Toc2248870)

[3.3.5.5 Federated Authentication Ready State 132](#_Toc2248871)

[3.3.5.6 Logged In State 133](#_Toc2248872)

[3.3.5.7 Client Request Execution State 133](#_Toc2248873)

[3.3.5.8 Routing Completed State 133](#_Toc2248874)

[3.3.5.9 Final State 134](#_Toc2248875)

[3.3.6 Timer Events 134](#_Toc2248876)

[3.3.7 Other Local Events 134](#_Toc2248877)

[4 Protocol Examples 135](#_Toc2248878)

[4.1 Pre-Login Request 135](#_Toc2248879)

[4.2 Login Request 136](#_Toc2248880)

[4.3 Login Request with Federated Authentication 138](#_Toc2248881)

[4.4 Login Response 145](#_Toc2248882)

[4.5 Login Response with Federated Authentication Feature Extension Acknowledgement 148](#_Toc2248883)

[4.6 SQL Batch Client Request 152](#_Toc2248884)

[4.7 SQL Batch Server Response 153](#_Toc2248885)

[4.8 RPC Client Request 155](#_Toc2248886)

[4.9 RPC Server Response 157](#_Toc2248887)

[4.10 Attention Request 158](#_Toc2248888)

[4.11 SSPI Message 158](#_Toc2248889)

[4.12 Bulk Load 159](#_Toc2248890)

[4.13 Transaction Manager Request 160](#_Toc2248891)

[4.14 TVP Insert Statement 161](#_Toc2248892)

[4.15 SparseColumn Select Statement 163](#_Toc2248893)

[4.16 FeatureExt with SESSIONRECOVERY Feature Data 168](#_Toc2248894)

[4.17 FeatureExtAck with SESSIONRECOVERY Feature Data 173](#_Toc2248895)

[4.18 Table Response with SESSIONSTATE Token Data 178](#_Toc2248896)

[4.19 Token Stream Communication 180](#_Toc2248897)

[4.19.1 Sending a SQL Batch 180](#_Toc2248898)

[4.19.2 Out-of-Band Attention Signal 180](#_Toc2248899)

[4.20 FeatureExt with AZURESQLSUPPORT Feature Data 181](#_Toc2248900)

[4.21 FeatureExtAck with AZURESQLSUPPORT Feature Data 184](#_Toc2248901)

[5 Security 192](#_Toc2248902)

[5.1 Security Considerations for Implementers 192](#_Toc2248903)

[5.2 Index of Security Parameters 192](#_Toc2248904)

[6 Appendix A: Product Behavior 193](#_Toc2248905)

[7 Change Tracking 199](#_Toc2248906)

[8 Index 200](#_Toc2248907)

# Introduction

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

* Authentication and channel encryption negotiation.
* Specification of requests in SQL (including Bulk Insert).
* Invocation of a [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) or user-defined function, also known as a [**remote procedure call (RPC)**](#gt_8a7f6700-8311-45bc-af10-82e10accd331).
* The return of data.
* [**Transaction manager**](#gt_4553803e-9d8d-407c-ad7d-9e65e01d6eb3) requests.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

## Glossary

This document uses the following terms:

**big-endian**: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

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

**common language runtime user-defined type (CLR UDT)**: A data type that is created and defined by the user on a database server that supports SQL by using a Microsoft .NET Framework common language runtime assembly.

**data stream**: A stream of data that corresponds to specific Tabular Data Stream (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.

**Distributed Transaction Coordinator (DTC)**: A Windows service that coordinates transactions across multiple resource managers, including databases. For more information, see [[MSDN-DTC]](https://go.microsoft.com/fwlink/?LinkId=89994).

**enclave**: A protected region of memory that is used only on the server side. This region is within the address space of SQL Server, and it acts as a trusted execution environment. Only code that runs within the enclave can access data within that enclave. Neither the data nor the code inside the enclave can be viewed from the outside, even with a debugger.

**enclave computations**: Locally enabled cryptographic operations and other operations in Transact-SQL queries on encrypted columns that are performed inside an enclave.

**federated authentication**: An authentication mechanism that allows a security token service (STS) in one trust domain to delegate user authentication to an identity provider in another trust domain, while generating a security token for the user, when there is a trust relationship between the two domains.

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

**Global Transactions**: A feature that allows users to execute transactions across multiple databases that are hosted in a shared service, such as Microsoft Azure SQL Database.

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

**interface**: A group of related function prototypes in a specific order, analogous to a C++ virtual interface. Multiple objects, of different object class, may implement the same interface. A derived interface may be created by adding methods after the end of an existing interface. In the Distributed Component Object Model (DCOM), all interfaces initially derive from IUnknown.

**little-endian**: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

**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]](https://go.microsoft.com/fwlink/?LinkId=213737).

**Multiple Active Result Sets (MARS)**: A feature in Microsoft SQL Server that allows applications to have more than one pending request per connection. For more information, see [[MSDN-MARS]](https://go.microsoft.com/fwlink/?LinkId=98459).

**nullable column**: A database table column that is allowed to contain no value for a given row.

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

**query notification**: A feature in SQL Server that allows the client to register for notification on changes to a given query result. For more information, see [[MSDN-QUERYNOTE]](https://go.microsoft.com/fwlink/?LinkId=119984).

**remote procedure call (RPC)**: A communication protocol used primarily between client and server. The term has three definitions that are often used interchangeably: a runtime environment providing for communication facilities between computers (the RPC runtime); a set of request-and-response message exchanges between computers (the RPC exchange); and the single message from an RPC exchange (the RPC message). For more information, see [[C706]](https://go.microsoft.com/fwlink/?LinkId=89824).

**result set**: A list of records that results from running a stored procedure or query, or applying a filter. The structure and content of the data in a result set varies according to the implementation.

**Security Support Provider Interface (SSPI)**: An API that allows connected applications to call one of several security providers to establish authenticated connections and to exchange data securely over those connections. It is equivalent to Generic Security Services (GSS)-API, and the two are on-the-wire compatible.

**Session Multiplex Protocol (SMP)**: A multiplexing protocol that enables multiple logical client connections to share a single transport connection to a server. Used by [**Multiple Active Result Sets (MARS)**](#gt_762fe1e3-0979-4402-b963-1e9150de133d). For more information, see [[MC-SMP]](%5bMC-SMP%5d.pdf#Section_04c8edde371d4af5bb33a39b3948f0af).

**Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)**: An authentication mechanism that allows Generic Security Services (GSS) peers to determine whether their credentials support a common set of GSS-API security mechanisms, to negotiate different options within a given security mechanism or different options from several security mechanisms, to select a service, and to establish a security context among themselves using that service. [**SPNEGO**](#gt_bc2f6b5e-e5c0-408b-8f55-0350c24b9838) is specified in [[RFC4178]](https://go.microsoft.com/fwlink/?LinkId=90461).

**SQL batch**: A set of [**SQL statements**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d).

**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]](https://go.microsoft.com/fwlink/?LinkId=213738).

**SQL Server User Authentication (SQLAUTH)**: An authentication mechanism that is 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 that the server understands.

**stored procedure**: A precompiled collection of SQL statements and, optionally, control-of-flow statements that are stored under a name and processed as a unit. They are stored in a SQL database and can be run with one call from an application. Stored procedures return an integer return code and can additionally return one or more result sets. Also referred to as sproc.

**table response**: A collection of data, all formatted in a specific manner, that is sent by the server to the client for the purpose of communicating the result of a client request. The server returns the result in a table response format for LOGIN7, SQL, and remote procedure call (RPC) requests.

**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.

**transaction manager**: The party that is responsible for managing and distributing the outcome of atomic transactions. A transaction manager is either a root transaction manager or a subordinate transaction manager for a specified transaction.

**Unicode**: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) standard [[UNICODE5.0.0/2007]](https://go.microsoft.com/fwlink/?LinkId=154659) provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

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

**MAY, SHOULD, MUST, SHOULD NOT, MUST NOT:** These terms (in all caps) are used as defined in [[RFC2119]](https://go.microsoft.com/fwlink/?LinkId=90317). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

## References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the [Errata](https://go.microsoft.com/fwlink/?linkid=850906).

### 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](mailto: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/service-names-port-numbers/service-names-port-numbers.xhtml](https://go.microsoft.com/fwlink/?LinkId=89888)

[IEEE754] IEEE, "IEEE Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, [http://ieeexplore.ieee.org/servlet/opac?punumber=2355](https://go.microsoft.com/fwlink/?LinkId=89903)

[IETF-AuthEncr] McGrew, D., Foley, J., and Paterson, K., "Authenticated Encryption with AES-CBC and HMAC-SHA", Network Working Group Internet-Draft, July 2014, [http://tools.ietf.org/html/draft-mcgrew-aead-aes-cbc-hmac-sha2-05](https://go.microsoft.com/fwlink/?LinkId=524322)

[MS-BINXML] Microsoft Corporation, "[SQL Server Binary XML Structure](%5bMS-BINXML%5d.pdf#Section_11ab6e8d247244d1a9e6bddf000e12f6)".

[MS-LCID] Microsoft Corporation, "[Windows Language Code Identifier (LCID) Reference](%5bMS-LCID%5d.pdf#Section_70feba9f294e491eb6eb56532684c37f)".

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989, [http://www.rfc-editor.org/rfc/rfc1122.txt](https://go.microsoft.com/fwlink/?LinkId=112180)

[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](https://go.microsoft.com/fwlink/?LinkId=90317)

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, [http://www.rfc-editor.org/rfc/rfc2246.txt](https://go.microsoft.com/fwlink/?LinkId=90324)

[RFC4234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, [http://www.rfc-editor.org/rfc/rfc4234.txt](https://go.microsoft.com/fwlink/?LinkId=90462)

[RFC6101] Freier, A., Karlton, P., and Kocher, P., "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, August 2011, [http://www.rfc-editor.org/rfc/rfc6101.txt](https://go.microsoft.com/fwlink/?LinkId=509953)

[RFC6234] Eastlake III, D., and Hansen, T., "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011, [http://www.rfc-editor.org/rfc/rfc6234.txt](https://go.microsoft.com/fwlink/?LinkId=328921)

[RFC793] Postel, J., Ed., "Transmission Control Protocol: DARPA Internet Program Protocol Specification", RFC 793, September 1981, [http://www.rfc-editor.org/rfc/rfc793.txt](https://go.microsoft.com/fwlink/?LinkId=150872)

[UNICODE] The Unicode Consortium, "The Unicode Consortium Home Page", [http://www.unicode.org/](https://go.microsoft.com/fwlink/?LinkId=90550)

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

### Informative References

[MC-SMP] Microsoft Corporation, "[Session Multiplex Protocol](%5bMC-SMP%5d.pdf#Section_04c8edde371d4af5bb33a39b3948f0af)".

[MS-NETOD] Microsoft Corporation, "[Microsoft .NET Framework Protocols Overview](%5bMS-NETOD%5d.pdf#Section_bcca8164da0843f2a983c34ed99171b0)".

[MS-SSCLRT] Microsoft Corporation, "[Microsoft SQL Server CLR Types Serialization Formats](%5bMS-SSCLRT%5d.pdf#Section_77460aa98c2f4449a65e1d649ebd77fa)".

[MSDN-Autocommit] Microsoft Corporation, "Autocommit Transactions", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms187878(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=145156)

[MSDN-BEGIN] Microsoft Corporation, "BEGIN TRANSACTION (Transact SQL)", [https://docs.microsoft.com/en-us/sql/t-sql/language-elements/begin-transaction-transact-sql](https://go.microsoft.com/fwlink/?LinkId=144544)

[MSDN-BOUND] Microsoft Corporation, "Using Bound Sessions", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms177480(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=144543)

[MSDN-BROWSE] Microsoft Corporation, "Browse Mode", in SQL Server 2000 Retired Technical documentation, p. 12261, [https://www.microsoft.com/en-us/download/confirmation.aspx?id=51958](https://go.microsoft.com/fwlink/?LinkId=140931)

[MSDN-Collation] Microsoft Corporation, "Collation and Unicode Support", [https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support](https://go.microsoft.com/fwlink/?LinkId=233327)

[MSDN-ColSets] Microsoft Corporation, "Use Column Sets", [https://docs.microsoft.com/en-us/sql/relational-databases/tables/use-column-sets](https://go.microsoft.com/fwlink/?LinkId=128616)

[MSDN-ColSortSty] Microsoft Corporation, "Windows Collation Sorting Styles", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms143515(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=233328)

[MSDN-COMMIT] Microsoft Corporation, "COMMIT TRANSACTION (Transact-SQL)", [https://docs.microsoft.com/en-us/sql/t-sql/language-elements/commit-transaction-transact-sql](https://go.microsoft.com/fwlink/?LinkId=144542)

[MSDN-DTC] Microsoft Corporation, "Distributed Transaction Coordinator", [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms684146(v=vs.85)](https://go.microsoft.com/fwlink/?LinkId=89994)

[MSDN-INSERT] Microsoft Corporation, "INSERT (Transact-SQL)", [https://docs.microsoft.com/en-us/sql/t-sql/statements/insert-transact-sql](https://go.microsoft.com/fwlink/?LinkId=154273)

[MSDN-ITrans] Microsoft Corporation, "ITransactionExport::GetTransactionCookie", [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms679869(v=vs.85)](https://go.microsoft.com/fwlink/?LinkId=146594)

[MSDN-MARS] Microsoft Corporation, "Using Multiple Active Result Sets (MARS)", [https://docs.microsoft.com/en-us/sql/relational-databases/native-client/features/using-multiple-active-result-sets-mars](https://go.microsoft.com/fwlink/?LinkId=98459)

[MSDN-MDAC] Wilkes, R., Bunch, A., and Dove, D., "Microsoft Data Access Components (MDAC) Installation", May 2005, [https://docs.microsoft.com/en-us/previous-versions/ms810805(v=msdn.10)](https://go.microsoft.com/fwlink/?LinkId=213737)

[MSDN-NamedPipes] Microsoft Corporation, "Creating a Valid Connection String Using Named Pipes", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms189307(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=127839)

[MSDN-NP] Microsoft Corporation, "Named Pipes", [https://docs.microsoft.com/en-us/windows/desktop/ipc/named-pipes](https://go.microsoft.com/fwlink/?LinkId=90247)

[MSDN-NTLM] Microsoft Corporation, "Microsoft NTLM", [https://docs.microsoft.com/en-us/windows/desktop/SecAuthN/microsoft-ntlm](https://go.microsoft.com/fwlink/?LinkId=145227)

[MSDN-QUERYNOTE] Microsoft Corporation, "Using Query Notifications", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms175110(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=119984)

[MSDN-SNAC] Microsoft Corporation, "Microsoft SQL Server Native Client and Microsoft SQL Server 2008 Native Client", [https://blogs.msdn.microsoft.com/sqlnativeclient/2008/02/27/microsoft-sql-server-native-client-and-microsoft-sql-server-2008-native-client/](https://go.microsoft.com/fwlink/?LinkId=213738)

[MSDN-SQLCollation] Microsoft Corporation, "Selecting a SQL Server Collation", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms144250(v=sql.105)](https://go.microsoft.com/fwlink/?LinkId=119987)

[MSDN-TDSENDPT] Microsoft Corporation, "Network Protocols and TDS Endpoints", [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms191220(v=sql.105)](https://go.microsoft.com/fwlink/?linkid=865399)

[MSDN-UPDATETEXT] Microsoft Corporation, "UPDATETEXT (Transact-SQL)", [https://docs.microsoft.com/en-us/sql/t-sql/queries/updatetext-transact-sql](https://go.microsoft.com/fwlink/?LinkId=154272)

[MSDN-WRITETEXT] Microsoft Corporation, "WRITETEXT (Transact-SQL)", [https://docs.microsoft.com/en-us/sql/t-sql/queries/writetext-transact-sql](https://go.microsoft.com/fwlink/?LinkId=154269)

[MSDOCS-DBMirror] Microsoft Corporation, "Database Mirroring in SQL Server", [https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/database-mirroring-in-sql-server](https://go.microsoft.com/fwlink/?linkid=874052)

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, [https://www.rfc-editor.org/rfc/rfc4120.txt](https://go.microsoft.com/fwlink/?LinkId=90458)

[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, [https://www.rfc-editor.org/rfc/rfc4178.txt](https://go.microsoft.com/fwlink/?LinkId=90461)

[SSPI] Microsoft Corporation, "SSPI", [https://docs.microsoft.com/en-us/windows/desktop/SecAuthN/sspi](https://go.microsoft.com/fwlink/?LinkId=90536)

## 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 has to be established. Note that the [**TDS session**](#gt_276cd76b-c0a2-4f7c-8529-ad0d60aa9592) 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**](#gt_1dfba466-175f-4050-a0e7-c1baf187d21d), [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) calls, returning data, and [**transaction manager**](#gt_4553803e-9d8d-407c-ad7d-9e65e01d6eb3) requests. Returned data is self-describing and record-oriented. The [**data streams**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e) describe the names, types and optional descriptions of the rows being returned. The following diagram depicts a (simplified) typical flow of communication in the TDS Protocol.

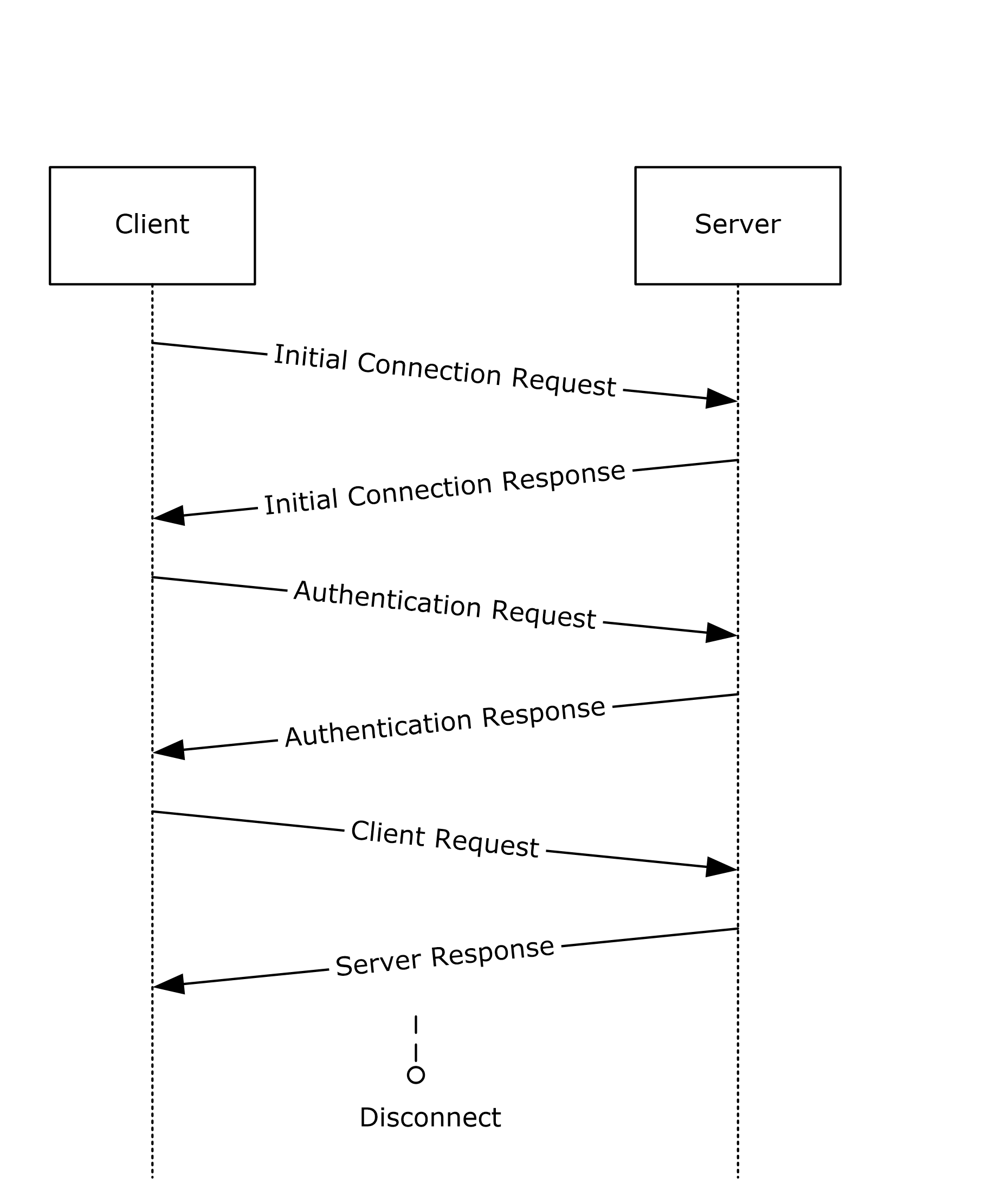


Figure 1: Communication flow in the TDS protocol

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**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d). It is assumed that the client and the server have already established a connection and authentication has succeeded.

1. 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.

1. Server:COLMETADATAdata stream
2. ROWdata stream
3. .
4. .
5. ROWdata stream
6. DONEdata stream

For more information about the correlation between data stream and TDS packet, see section [2.2.4](#Section_dc3a08548230482fbbb9d94a3b905a26).[<1>](#Appendix_A_1" \o "Product behavior note 1)

Additional details about which SQL Server version corresponds to which TDS version number are defined in LOGINACK (section [2.2.7.13](#Section_490e563dcc6e4c86bb95ef0186b98032)).

## 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)**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) feature [[MSDN-MARS]](https://go.microsoft.com/fwlink/?LinkId=98459) is enabled, then the [**Session Multiplex Protocol (SMP)**](#gt_f70f98cc-c555-4a40-9509-bc1da4021211) [[MC-SMP]](%5bMC-SMP%5d.pdf#Section_04c8edde371d4af5bb33a39b3948f0af) is required.

This relationship is illustrated in the following figure.

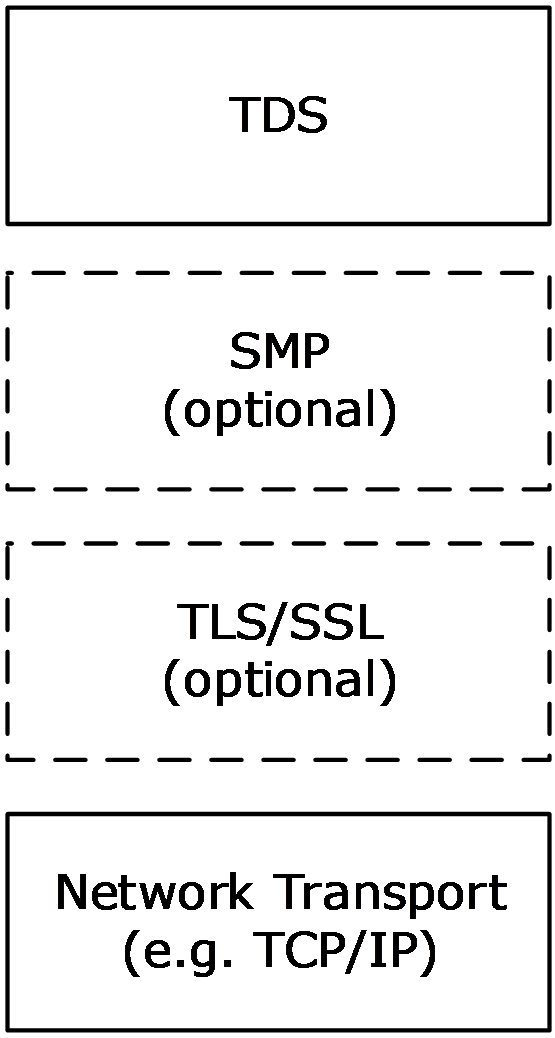


Figure 2: Protocol relationship

## 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)**](#gt_fb216516-748b-4873-8bdd-64c5f4da9920) [[SSPI]](https://go.microsoft.com/fwlink/?LinkId=90536) authentication to be used, SSPI support needs to be available on both the client and server machines. For channel encryption to be used, TLS/SSL support needs to be present on both client and server machines, and a certificate suitable for encryption has to be deployed on the server machine. For [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) to be used, a library that provides federated authentication support or an equivalent needs to be present on the server, and the client needs to be able to generate a token for federated authentication.

## 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.

## 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](#Section_fd30432f71b2488cb30f19737d76d970).
* **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 [2.2.6.4](#Section_773a62b6ee894c029e5e344882630aac).

**Note**  After a protocol feature is introduced, subsequent versions of the TDS protocol support that feature until that feature is removed.

* **Security and Authentication Methods:** The TDS protocol supports [**SQL Server User Authentication (SQLAUTH)**](#gt_1a186995-43e5-4e46-896a-bad208ec2551). 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 2.2.6.4 and [3.2.5.1](#Section_cc823ca848674387819dcc5c19da5732). The TDS protocol also supports [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19). The use of federated authentication in TDS is defined in sections 2.2.6.4 and [3.2.5](#Section_bd3a16dd75b64546933059c3ef44d50e).
* **Localization:** Localization-dependent protocol behavior is specified in sections [2.2.5.1.2](#Section_3d29e8dc218a42c69ba4947ebca9fd7e) and [2.2.5.6](#Section_cbe9c510eae64b1f9893a098944d430a).
* **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**](#gt_276cd76b-c0a2-4f7c-8529-ad0d60aa9592) 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 [2.2.6.5](#Section_60f5640801884cd58b9025c6f2423868).

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]](https://go.microsoft.com/fwlink/?LinkId=90324) and [[RFC6101]](https://go.microsoft.com/fwlink/?LinkId=509953).

## Vendor-Extensible Fields

None.

## Standards Assignments

| Parameter | TCP port value | Reference |
| --- | --- | --- |
| Default SQL Server instance TCP port | 1433 | [[IANAPORT]](https://go.microsoft.com/fwlink/?LinkId=89888) |

# Messages

The formal syntax of all messages is provided in Augmented Backus-Naur Form (ABNF) [[RFC4234]](https://go.microsoft.com/fwlink/?LinkId=90462), with the addition of the following:

* underscore ("\_") – a valid character within an identifier.
* "%x00" – a valid value.
* "%b0" – a valid value.

## 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>](#Appendix_A_2" \o "Product behavior note 2)

* TCP [[RFC793]](https://go.microsoft.com/fwlink/?LinkId=150872).
* A reliable transport over the [**Virtual Interface Architecture (VIA)**](#gt_c35909bd-185e-4b60-be82-995a0318873e) [**interface**](#gt_95913fbd-3262-47ae-b5eb-18e6806824b9) [VIA2002].[<3>](#Appendix_A_3" \o "Product behavior note 3)
* Named Pipes [[MSDN-NP]](https://go.microsoft.com/fwlink/?LinkId=90247).
* Shared memory [[MSDN-TDSENDPT]](https://go.microsoft.com/fwlink/?linkid=865399).
* Optionally, the TDS protocol has implementations for the following two protocols on top of the preceding transports:
  + Transport Layer Security (TLS) [[RFC2246]](https://go.microsoft.com/fwlink/?LinkId=90324)/Secure Socket Layer (SSL), in case TLS/SSL encryption is negotiated.
  + [**Session Multiplex Protocol (SMP)**](#gt_f70f98cc-c555-4a40-9509-bc1da4021211) [[MC-SMP]](%5bMC-SMP%5d.pdf#Section_04c8edde371d4af5bb33a39b3948f0af), in case the [**Multiple Active Result Sets (MARS)**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) feature [[MSDN-MARS]](https://go.microsoft.com/fwlink/?LinkId=98459) is requested.

## Message Syntax

Character data, such as [**SQL statements**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d), within a TDS message is in [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8), unless the character data represents the data value of an ASCII data type, such as a non-Unicode data column. A character count within TDS is a count of characters, rather than of bytes, except when that character count is explicitly specified as a byte count.

### Client Messages

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

* [Pre-Login](#Section_58886b79ec7542f3bd7be32f327366b6)
* [Login](#Section_f159ec89ef1a4fcd9ffd8ee701fccbb0)
* [Federated Authentication Token](#Section_f40cca2929b843ecb9ff2f8682486c29)
* [SQL Batch](#Section_5b7f9062d5144b4ea7c59a27a19596c4)
* [Bulk Load](#Section_88176081df754b24bcfb4c16ff03cbfa)
* [Remote Procedure Call](#Section_26327437aa3c4e969bba73a6e862ba21)
* [Attention](#Section_dc28579f49b14a789c5f63fbda002d2e)
* [Transaction Manager Request](#Section_b4b7856454404fc0b5efc9e1925aaefe)

These messages are briefly described in the sections that follow. Detailed descriptions of message contents are in section [2.2.6](#Section_c060af9c6db74360954cc28a132c8949).

#### Pre-Login

Before a login occurs, a Pre-Login handshake occurs between client and server, setting up contexts such as encryption and MARS-enabled. For more details, see section [2.2.6.5](#Section_60f5640801884cd58b9025c6f2423868).

#### 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 connection is established separately in the same way. For more details, see section [2.2.6.4](#Section_773a62b6ee894c029e5e344882630aac).

After the server receives the login record from the client and, if necessary, performs subsequent authentication handshakes (such as when SSPI [[SSPI]](https://go.microsoft.com/fwlink/?LinkId=90536) or [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) is used), the server notifies the client that it has either accepted or rejected the connection request. For more details, see section [3.3.5.1](#Section_8bfcbcd21baf47928fff86a63a9e90fa).

#### Federated Authentication Token

When the client indicates in the Login record that [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19)[<4>](#Appendix_A_4" \o "Product behavior note 4) is to be used but that the intended client library needs additional information from the server to generate a federated authentication token, if the server supports federated authentication that uses that client library, the server responds with a token that the client uses to perform federated authentication. The client then generates and sends a tokenless Federated Authentication Token message that contains binary authentication data that is generated by the federated authentication library. For more details, see section [2.2.6.3](#Section_827d963229574d54b9ea384530ae79d0).

After the server receives the Federated Authentication Token message from the client, the server notifies the client that it has either accepted or rejected the connection request. For more details, see section 3.3.5.

#### SQL Batch

To send a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) or a batch of SQL statements, the [**SQL batch**](#gt_1dfba466-175f-4050-a0e7-c1baf187d21d), represented by a [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) 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 [2.2.6.7](#Section_f2026cd39a464a3f9a08f63140bcbbe3).

#### Bulk Load

In a [**bulk insert**](#gt_81e2b338-0f2a-4980-9824-9c27bb6d1341)/bulk load operation, a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) consists of a [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) string that is followed by binary data. The client sends the **INSERT BULK** SQL statement and then sends a [COLMETADATA](#Section_58880b9f381c43b2bf8b0727a98c4f4c) token that describes the raw data. 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](#Section_ab4a7d62cd1f4db1b67decae58f493e3).

A bulk load operation is also used for inserting data with a previously issued **UPDATETEXT BULK** or **WRITETEXT BULK** SQL statement. For more details, see section [2.2.6.2](#Section_6ee5bed798d844f992177b10e2123596).

#### Remote Procedure Call

To execute a [**remote procedure call (RPC)**](#gt_8a7f6700-8311-45bc-af10-82e10accd331) on the server, the client sends an RPC message [**data stream**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e) 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**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d). There can be several RPCs in one message. For more details, see section [2.2.6.6](#Section_619c43b694954a589e49a4950db245b3).

#### Attention

The client can interrupt and cancel the current request by sending an **Attention** message. This is also known as [**out-of-band**](#gt_26c1caf3-c889-4b99-a22b-9da056d397cf) 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](#Section_f8aa004ebf764f94bd5b39597a888146).

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 bits terminates the current request, and the server MUST ignore the current request. When the ignore and EOM bits are set, the server does not send an attention acknowledgment, but instead returns a [**table response**](#gt_71dd1dd2-c167-49a8-a5f5-6b0df5c8b48a) with a single DONE token that has a status of DONE\_ERROR to indicate that the incoming request was ignored. For more details about the packet header status code, see section [2.2.3.1.2](#Section_ce398f9a7d474ede8f369dd6fc21ca43).

#### Transaction Manager Request

The client can request that the connection enlist in a transaction as described in [[MSDN-DTC]](https://go.microsoft.com/fwlink/?LinkId=89994).

### Server Messages

Messages sent from the server to the client are the following:

* [Pre-Login Response](#Section_63c37a5d459d40b98c4a47e13aff235d)
* [Login Response](#Section_da5a73bbe49247ab9d9c23ac00292efc)
* [Federated Authentication Information](#Section_1a22145b7657435d9c60318be82456e1)
* [Row Data](#Section_cc5aaa57f2e04571910ac5f6a538f88f)
* [Return Status](#Section_ed493f4cf63d41739476e64261400119)
* [Return Parameters](#Section_43acbd448d754f2e8f5c543b59a9d517)
* [Response Completion](#Section_6ee2125f34be43c6a38ce9ee7338500f)
* [Error and Info](#Section_7fffdf0c78d140fd9dee1e6b4246d3aa)
* [Attention Acknowledgement](#Section_9bcc4b79a79141d0bf7673a97e501b35)

These messages are briefly described in the sections that follow. Detailed descriptions of message contents are in section [2.2.6](#Section_c060af9c6db74360954cc28a132c8949) and section [2.2.7](#Section_67b6113cd72242d1902c3f6e8de09173).

#### Pre-Login Response

The Pre-Login Response message is a tokenless packet data stream. The data stream consists of the response to the information requested by the client's Pre-Login message. For more details, see section [2.2.6.5](#Section_60f5640801884cd58b9025c6f2423868).

#### Login Response

The Login Response message is a token stream that consists of information about the server's characteristics, optional information and error messages, and finally, a completion message.

The LOGINACK token [**data stream**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e) includes information about the server [**interface**](#gt_95913fbd-3262-47ae-b5eb-18e6806824b9) and the server's product code and name. For more details, see section [2.2.7.13](#Section_490e563dcc6e4c86bb95ef0186b98032).

If there are any messages in the login response, an [ERROR](#Section_9805e9fa1f8b4cf88f788d2602228635) or [INFO](#Section_284bb815d0834ed5b33abdc2492e322b) token data stream is returned from the server to the client. For more details, see sections 2.2.7.9 and 2.2.7.12.

The server can send, as part of the login response, one or more [ENVCHANGE](#Section_2b3eb7e5d43d4d1bbf4d76b9e3afc791) 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 more details, see section 2.2.7.8.

A done packet MUST be present as the final part of the login response, and a [DONE](#Section_3c06f11098bd4d5bb836b1ba66452cb7) token data stream is the last thing sent in response to a server login request. For more details, see section 2.2.7.5.

#### Federated Authentication Information

After the server receives a Login message that states that the client intends to use a [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) token from a specific client library that needs additional information from the server to generate that token, if the server supports federated authentication that uses that client library, the server responds to the client with a message. This message contains a Federated Authentication Information Token that provides the information necessary for the client to generate a federated authentication token. If the server determines that no information is required for this particular client library, the server does not send the information token. For more details, see section [2.2.7.11](#Section_0e4486d6d407496298030c1a4d4d87ce).

#### 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](#Section_58880b9f381c43b2bf8b0727a98c4f4c).

#### Return Status

When a [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) is executed by the server, the server MUST return a status value. This is a 4-byte integer and is sent via the [RETURNSTATUS](#Section_c719f199e71b418790b994f78bd1870e) token. A stored procedure execution is requested through either an RPC Batch or a [SQL Batch](#Section_5b7f9062d5144b4ea7c59a27a19596c4) message. For more information, see section 2.2.7.17.

#### Return Parameters

The response format for execution of a [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) is identical regardless of whether the request was sent as [SQL Batch](#Section_5b7f9062d5144b4ea7c59a27a19596c4) 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](#Section_7091f6f6b83d4ed2afebba5013dfb18f) 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 2.2.7.18.

#### Response Completion

The client reads results in logical units and can tell when all results have been received by examining the [DONE](#Section_3c06f11098bd4d5bb836b1ba66452cb7) token [**data stream**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e).

When executing a batch of [**SQL statements**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d), 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, see section 2.2.7.5.

For [**stored procedures**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7), completion of SQL statements in the stored procedure is indicated by a [DONEINPROC](#Section_43e891c5f7a1432f8f9f233c4cd96afb) token data stream for each SQL statement and a [DONEPROC](#Section_65e24140edea46e5b710209af2016195) 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.

#### Error and Info

Besides returning descriptions of Row data and the data itself, TDS provides a token [**data stream**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e) type for the server to send error and informational messages to the client. These are the [ERROR](#Section_9805e9fa1f8b4cf88f788d2602228635) token data stream and the [INFO](#Section_284bb815d0834ed5b33abdc2492e322b) token data stream. For more details, see section 2.2.7.9 and section 2.2.7.12, respectively.

#### 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. Attention messages are acknowledged in the [DONE](#Section_3c06f11098bd4d5bb836b1ba66452cb7) token data stream. For more details, see section 2.2.7.5.

### 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](#Section_773a62b6ee894c029e5e344882630aac) 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](#Section_2b3eb7e5d43d4d1bbf4d76b9e3afc791) 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](#Section_c1cddd03b448470a946a9b1b908f27a7).

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.

#### 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 packet header states the **Type** and **Length** of the entire packet.

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

##### Type

**Type** defines the type of message. **Type** is a 1-byte unsigned char. The following table describes the types that are available.

| Value | Description | Packet contains data? |
| --- | --- | --- |
| 1 | SQL batch. | Yes |
| 2 | Pre-TDS7 Login[<5>](#Appendix_A_5" \o "Product behavior note 5) | Yes |
| 3 | RPC | Yes |
| 4 | Tabular result | Yes |
| 5 | Unused |  |
| 6 | Attention signal | No |
| 7 | Bulk load data | Yes |
| 8 | Federated Authentication Token | Yes |
| 9-13 | Unused |  |
| 14 | Transaction manager request | Yes |
| 15 | Unused |  |
| 16 | TDS7 Login[<6>](#Appendix_A_6" \o "Product behavior note 6) | Yes |
| 17 | SSPI | Yes |
| 18 | Pre-Login | 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](#Section_52b5da79b22248fdb408dd4c67464f9a)), 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](#Section_7ea9ee1ab46141f29004141c0e712935) and [2.2.2](#Section_342f4cbb2b4b489c8b63f99b12021a94), correspond to which packet header type.

| Message type | Client or server message | Packet header type |
| --- | --- | --- |
| [Pre-Login](#Section_58886b79ec7542f3bd7be32f327366b6) | Client | 2 or 18 depending on whether the client supports TDS v7.0+ |
| [Login](#Section_f159ec89ef1a4fcd9ffd8ee701fccbb0) | Client | 16 + 17 (if Integrated authentication) |
| [Federated Authentication Token](#Section_827d963229574d54b9ea384530ae79d0) | Client | 8 |
| [SQL Batch](#Section_5b7f9062d5144b4ea7c59a27a19596c4) | Client | 1 |
| [Bulk Load](#Section_88176081df754b24bcfb4c16ff03cbfa) | Client | 7 |
| [RPC](#Section_26327437aa3c4e969bba73a6e862ba21) | Client | 3 |
| [Attention](#Section_dc28579f49b14a789c5f63fbda002d2e) | Client | 6 |
| [Transaction Manager Request](#Section_b4b7856454404fc0b5efc9e1925aaefe) | Client | 14 |
| [FeatureExtAck](#Section_2eb82f8e11f046dcb42d27302fa4701a) | Server | 4 |
| [Pre-Login Response](#Section_63c37a5d459d40b98c4a47e13aff235d) | Server | 4 |
| [Login Response](#Section_da5a73bbe49247ab9d9c23ac00292efc) | Server | 4 |
| [Federated Authentication Information](#Section_0e4486d6d407496298030c1a4d4d87ce) | Server | 4 |
| [Row Data](#Section_cc5aaa57f2e04571910ac5f6a538f88f) | Server | 4 |
| [Return Status](#Section_ed493f4cf63d41739476e64261400119) | Server | 4 |
| [Return Parameters](#Section_43acbd448d754f2e8f5c543b59a9d517) | Server | 4 |
| [Response Completion](#Section_6ee2125f34be43c6a38ce9ee7338500f) | Server | 4 |
| [Session State](#Section_626fbe19f3564599ba17c70f44005106) | Server | 4 |
| [Error and Info](#Section_7fffdf0c78d140fd9dee1e6b4246d3aa) | Server | 4 |
| [Attention Acknowledgement](#Section_9bcc4b79a79141d0bf7673a97e501b35) | Server | 4 |

##### 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 Transaction Manager 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.

##### 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**](#gt_6f6f9e8e-5966-4727-8527-7e02fb864e7e)). 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.

##### 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 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**](#gt_6f6f9e8e-5966-4727-8527-7e02fb864e7e)).

##### 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.[<7>](#Appendix_A_7" \o "Product behavior note 7) This allows the receiver to track the sequence of TDS packets for a given message. This value is currently ignored.

##### Window

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

#### Packet Data

Packet data for a given message follows the packet header (see **Type** in section [2.2.3.1.1](#Section_9b4a463c26344a4bac35bebfff2fb0f7) 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](#Section_ce398f9a7d474ede8f369dd6fc21ca43) 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](#Section_ec9e8663191c4dd1baa848bbfba5ed7e).

### 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 that is used to describe the data that follows it and contains information such as 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](#Section_7ea9ee1ab46141f29004141c0e712935) and [2.2.2](#Section_342f4cbb2b4b489c8b63f99b12021a94), use token streams and which do not.

| Message type | Client or server message | Token stream? |
| --- | --- | --- |
| Pre-Login | Client | No |
| Login | Client | No |
| Federated Authentication Token | Client | No |
| SQL Command | Client | No |
| Bulk Load | Client | Yes |
| Remote Procedure Call (RPC) | Client | Yes |
| Attention | Client | No |
| Transaction Manager Request | Client | No |
| Pre-Login Response | Server | No |
| Federated Authentication Information | Server | Yes |
| FeatureExtAck | Server | Yes |
| Login Response | Server | Yes |
| Row Data | Server | Yes |
| Return Status | Server | Yes |
| Return Parameters | Server | Yes |
| Response Completion | Server | Yes |
| Session State | Server | Yes |
| Error and Info | Server | Yes |
| Attention Acknowledgement | Server | No |

#### Tokenless Stream

As shown in the previous section, some messages do not use tokens to describe the data portion of the [**data stream**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e). 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.

#### 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](#Section_67b6113cd72242d1902c3f6e8de09173).

##### Token Definition

There are four classes of token definitions:

* [Zero Length Token(xx01xxxx)](#Section_9b571ae55d0048748cf798f30ca69dbd)
* [Fixed Length Token(xx11xxxx)](#Section_5d73136323384ebeb1eb5cce0780ef8f)
* [Variable Length Tokens(xx10xxxx)](#Section_d3edea23be1f416098f50de233ffeebc)
* [Variable Count Tokens(xx00xxxx)](#Section_5ff30ab3aaa7474ba095662fcaed5549)

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).

###### 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:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 or 1 | 0 or 1 | 0 | 1 | 0 or 1 | 0 or 1 | 0 or 1 | 0 or 1 |

A value of “0 or 1” denotes a bit position that can contain the bit value “0” or “1”.

###### 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. Also in the table, a value of “0 or 1” denotes a bit position that can contain the bit value “0” or “1”.

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

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 [2.2.5.4.1](#Section_859eb3d280d340f6a637414552c9c552).

###### 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](#Section_aa8466c5ca3d48caa6387c1becebe754) 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.

###### 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. As shown in the following table, a variable count token always has its third and fourth bits set to “0”, and a value of “0 or 1” in the remaining bit positions denotes a bit position that can contain the bit value “0” or “1”.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 or 1 | 0 or 1 | 0 | 0 | 0 or 1 | 0 or 1 | 0 or 1 | 0 or 1 |

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

#### Done and Attention Tokens

The DONE token marks the end of the response for each executed [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d). 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**](#gt_26c1caf3-c889-4b99-a22b-9da056d397cf) 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](#Section_626fbe19f3564599ba17c70f44005106) data, from the server until the final DONE token, which acknowledges that the attention signal is read.[<8>](#Appendix_A_8" \o "Product behavior note 8)

### 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 by using Augmented Backus-Naur Form (ABNF) [[RFC4234]](https://go.microsoft.com/fwlink/?LinkId=90462). Details can be found in the following subsections.

#### 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.

1. BIT = %b0 / %b1

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

1. BYTE = 8BIT

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

1. BYTELEN = BYTE

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

1. USHORT = 2BYTE

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

1. LONG = 4BYTE

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

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.

1. DWORD = 32BIT

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

1. LONGLONG = 8BYTE

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

1. ULONGLONG = 8BYTE

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

1. UCHAR = BYTE

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

1. 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.

1. USHORTCHARBINLEN = 2BYTE

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

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.

1. ULONGLONGLEN = 8BYTE

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

1. PRECISION = 8BIT

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

1. SCALE = 8BIT

**GEN\_NULL**: A single byte (8-bit) value representing a NULL value.

1. 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](#Section_3f983fde0509485a8c40a9fa6679a828)) for additional details.

1. 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.

1. 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.

1. FRESERVEDBYTE = %x00

**UNICODECHAR**: A single [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) character in UCS-2 encoding, as specified in Unicode [[UNICODE]](https://go.microsoft.com/fwlink/?LinkId=90550).

1. UNICODECHAR = 2BYTE

**Notes**

* All integer types are represented in reverse byte order ([**little-endian**](#gt_079478cb-f4c5-4ce5-b72b-2144da5d2ce7)) 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.

##### 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:

1. 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**](#gt_079478cb-f4c5-4ce5-b72b-2144da5d2ce7) byte ordering. For example, the following rule definition:

1. 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.

##### Collation Rule Definition

The collation rule is used to specify collation information for character data or metadata describing character data.[<9>](#Appendix_A_9" \o "Product behavior note 9) This is typically specified as part of the [LOGIN7](#Section_773a62b6ee894c029e5e344882630aac) message or part of a column definition in server results containing character data. For more information about column definition, see [COLMETADATA](#Section_58880b9f381c43b2bf8b0727a98c4f4c).

1. LCID = 20BIT
2. fIgnoreCase = BIT
3. fIgnoreAccent = BIT
4. fIgnoreWidth = BIT
5. fIgnoreKana = BIT
6. fBinary = BIT
7. fBinary2 = BIT
8. ColFlags = fIgnoreCase fIgnoreAccent fIgnoreKana
9. fIgnoreWidth fBinary fBinary2 FRESERVEDBIT
10. FRESERVEDBIT
11. Version = 4BIT
12. SortId = BYTE
13. COLLATION = LCID ColFlags Version SortId

A SQL collation is one of a predefined set of sort orders. The sort orders are identified with non-zero SortId values described by [[MSDN-SQLCollation]](https://go.microsoft.com/fwlink/?LinkId=119987).

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]](%5bMS-LCID%5d.pdf#Section_70feba9f294e491eb6eb56532684c37f).

**Note**

* ColFlags is represented in [least significant bit order](#Section_bbc22f15e1a04338a169c79819d39b1c).
* A COLLATION[<10>](#Appendix_A_10" \o "Product behavior note 10) value of 0x00 00 00 00 00 specifies a request for the use of raw collation.

#### Data Stream Types

##### 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](#Section_7af536671b72470382587984e838f746).

1. BYTESTREAM = \*BYTE
2. UNICODESTREAM = \*(2BYTE)

##### 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.

1. B\_VARCHAR = BYTELEN \*CHAR
2. US\_VARCHAR = USHORTLEN \*CHAR

Note that the lengths of B\_VARCHAR and US\_VARCHAR are given in [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) 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.

1. B\_VARBYTE = BYTELEN \*BYTE
2. US\_VARBYTE = USHORTLEN \*BYTE
3. L\_VARBYTE = LONGLEN \*BYTE

##### 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.

1. TYPE\_VARLEN = BYTELEN
2. /
3. USHORTCHARBINLEN
4. /
5. 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 are read because "INT2TYPE" is always 2 bytes in length. For more details, see [Data Types Definitions](#Section_ffb02215af074b5085451fd522106c68).

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:

1. TYPE\_VARBYTE = GEN\_NULL / CHARBIN\_NULL / PLP\_BODY
2. / ([TYPE\_VARLEN] \*BYTE)

**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):

1. PLP\_BODY= PLP\_NULL
2. /
3. ((ULONGLONGLEN / UNKNOWN\_PLP\_LEN)
4. \*PLP\_CHUNK PLP\_TERMINATOR)
5. PLP\_NULL = %xFFFFFFFFFFFFFFFF
6. UNKNOWN\_PLP\_LEN = %xFFFFFFFFFFFFFFFE
7. PLP\_CHUNK = ULONGLEN 1\*BYTE
8. PLP\_TERMINATOR = %x00000000

**Notes**

* TYPE\_INFO rule specifies a Partially Length-prefixed Data type (PARTLENTYPE, see [2.2.5.4.3](#Section_7d26a257083e409b81ba897e0c672be0)).
* 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.

#### 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](#Section_e168d373a7b741aab6ca25985466a7e0), and the [Transaction Descriptor header](#Section_4257dd95ef6c4621b75d270738487d68) were introduced in TDS 7.2. The [Trace Activity header](#Section_6e9f106bdf6e4cbea6eb45ceb10c63be) 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 |
| Transaction Descriptor | 0x00 02 | Required | Required | Required |
| Trace Activity | 0x00 03 | Optional | Optional | Optional |

**Stream-Specific Rules:**

1. TotalLength = DWORD ;including itself
2. HeaderLength = DWORD ;including itself
3. HeaderType = USHORT;
4. HeaderData = \*BYTE
5. Header = HeaderLength HeaderType HeaderData

**Stream Definition:**

1. 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. |

##### Query Notifications Header

This packet data stream header allows the client to specify that a notification is to be supplied 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 notifications**](#gt_62a2f252-bd68-4d77-a751-d6ff27010678)[<11>](#Appendix_A_11" \o "Product behavior note 11) functionality for a database server that supports SQL, see [[MSDN-QUERYNOTE]](https://go.microsoft.com/fwlink/?LinkId=119984).

**Stream Specific Rules:**

1. NotifyId = USHORT UNICODESTREAM ; user specified value
2. ; when subscribing to
3. ; query notifications
4. SSBDeployment = USHORT UNICODESTREAM
5. NotifyTimeout = ULONG ; duration in which the query
6. ; notification subscription
7. ; 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.[<12>](#Appendix_A_12" \o "Product behavior note 12) The time unit of NotifyTimeout is milliseconds.

**Stream Definition:**

1. HeaderData = NotifyId
2. SSBDeployment
3. [NotifyTimeout]

##### Transaction Descriptor Header

This packet data stream contains information regarding transaction descriptor and number of outstanding requests as they apply to [**Multiple Active Result Sets (MARS)**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) [[MSDN-MARS]](https://go.microsoft.com/fwlink/?LinkId=98459).

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]](https://go.microsoft.com/fwlink/?LinkId=145156).

**Stream-Specific Rules:**

1. OutstandingRequestCount = DWORD ; number of requests currently active on
2. ; the connection
3. TransactionDescriptor = ULONGLONG ; for each connection, a number that uniquely
4. ; identifies the transaction with which the
5. ; request is associated; initially generated
6. ; by the server when a new transaction is
7. ; created and returned to the client as part
8. ; of the ENVCHANGE token stream

For more information about processing the Transaction Descriptor header, see section [2.2.6.9](#Section_0fb28ba5ddcb4d0295c3aa5b05ec6092).

**Stream Definition:**

1. HeaderData = TransactionDescriptor
2. OutstandingRequestCount

##### 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:**

1. GUID\_ActivityID = 16bytes ; client application activity id
2. ; used for debugging purposes
3. ActivitySequence = ULONG ; client application activity sequence
4. ; used for debugging purposes

**Stream Definition:**

1. HeaderData = ActivityId

#### 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.

##### Fixed-Length Data Types

The fixed-length data types include the following types.

1. NULLTYPE = %x1F ; Null
2. INT1TYPE = %x30 ; TinyInt
3. BITTYPE = %x32 ; Bit
4. INT2TYPE = %x34 ; SmallInt
5. INT4TYPE = %x38 ; Int
6. DATETIM4TYPE = %x3A ; SmallDateTime
7. FLT4TYPE = %x3B ; Real
8. MONEYTYPE = %x3C ; Money
9. DATETIMETYPE = %x3D ; DateTime
10. FLT8TYPE = %x3E ; Float
11. MONEY4TYPE = %x7A ; SmallMoney
12. INT8TYPE = %x7F ; BigInt
13. FIXEDLENTYPE = NULLTYPE
14. /
15. INT1TYPE
16. /
17. BITTYPE
18. /
19. INT2TYPE
20. /
21. INT4TYPE
22. /
23. DATETIM4TYPE
24. /
25. FLT4TYPE
26. /
27. MONEYTYPE
28. /
29. DATETIMETYPE
30. /
31. FLT8TYPE
32. /
33. MONEY4TYPE
34. /
35. INT8TYPE

Non-nullable values are returned using these fixed-length data types. There is no data associated with NULLTYPE.[<13>](#Appendix_A_13" \o "Product behavior note 13) 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 data to be followed. The SQL data types of the corresponding fixed-length data types are in the comment part of each data type.

##### 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.

1. GUIDTYPE = %x24 ; UniqueIdentifier
2. INTNTYPE = %x26 ; (see below)
3. DECIMALTYPE = %x37 ; Decimal (legacy support)
4. NUMERICTYPE = %x3F ; Numeric (legacy support)
5. BITNTYPE = %x68 ; (see below)
6. DECIMALNTYPE = %x6A ; Decimal
7. NUMERICNTYPE = %x6C ; Numeric
8. FLTNTYPE = %x6D ; (see below)
9. MONEYNTYPE = %x6E ; (see below)
10. DATETIMNTYPE = %x6F ; (see below)
11. DATENTYPE = %x28 ; (introduced in TDS 7.3)
12. TIMENTYPE = %x29 ; (introduced in TDS 7.3)
13. DATETIME2NTYPE = %x2A ; (introduced in TDS 7.3)
14. DATETIMEOFFSETNTYPE = %x2B ; (introduced in TDS 7.3)
15. CHARTYPE = %x2F ; Char (legacy support)
16. VARCHARTYPE = %x27 ; VarChar (legacy support)
17. BINARYTYPE = %x2D ; Binary (legacy support)
18. VARBINARYTYPE = %x25 ; VarBinary (legacy support)
19. BIGVARBINTYPE = %xA5 ; VarBinary
20. BIGVARCHRTYPE = %xA7 ; VarChar
21. BIGBINARYTYPE = %xAD ; Binary
22. BIGCHARTYPE = %xAF ; Char
23. NVARCHARTYPE = %xE7 ; NVarChar
24. NCHARTYPE = %xEF ; NChar
25. XMLTYPE = %xF1 ; XML (introduced in TDS 7.2)
26. UDTTYPE = %xF0 ; CLR UDT (introduced in TDS 7.2)
27. TEXTTYPE = %x23 ; Text
28. IMAGETYPE = %x22 ; Image
29. NTEXTTYPE = %x63 ; NText
30. SSVARIANTTYPE = %x62 ; Sql\_Variant (introduced in TDS 7.2)
31. BYTELEN\_TYPE = GUIDTYPE
32. /
33. INTNTYPE
34. /
35. DECIMALTYPE
36. /
37. NUMERICTYPE
38. /
39. BITNTYPE
40. /
41. DECIMALNTYPE
42. /
43. NUMERICNTYPE
44. /
45. FLTNTYPE
46. /
47. MONEYNTYPE
48. /
49. DATETIMNTYPE
50. /
51. DATENTYPE
52. /
53. TIMENTYPE
54. /
55. DATETIME2NTYPE
56. /
57. DATETIMEOFFSETNTYPE
58. /
59. CHARTYPE
60. /
61. VARCHARTYPE
62. /
63. BINARYTYPE
64. /
65. VARBINARYTYPE ; the length value associated
66. ; with these data types is
67. ; 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:

| 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.

1. USHORTLEN\_TYPE = BIGVARBINTYPE
2. /
3. BIGVARCHRTYPE
4. /
5. BIGBINARYTYPE
6. /
7. BIGCHARTYPE
8. /
9. NVARCHARTYPE
10. /
11. NCHARTYPE ; the length value associated with
12. ; these data types is specified
13. ; within a USHORT
14. LONGLEN\_TYPE = IMAGETYPE
15. /
16. NTEXTTYPE
17. /
18. SSVARIANTTYPE
19. /
20. TEXTTYPE
21. /
22. XMLTYPE ; the length value associated with
23. ; these data types is specified
24. ; within a LONG

**Notes**

* MaxLength for an SSVARIANTTYPE is 8009 (8000 for strings). For more details, see section [2.2.5.5.4](#Section_2435e85d9e61492cacb2627ffccb5b92).
* XMLTYPE is only a valid LONGLEN\_TYPE for BulkLoadBCP.

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

1. VARLENTYPE = BYTELEN\_TYPE
2. /
3. USHORTLEN\_TYPE
4. /
5. 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.

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.

##### 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](#Section_3f983fde0509485a8c40a9fa6679a828)).

1. PARTLENTYPE = XMLTYPE
2. /
3. BIGVARCHRTYPE
4. /
5. BIGVARBINTYPE
6. /
7. NVARCHARTYPE
8. /
9. 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.

#### Data Type Details

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

##### 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.

###### Integers

All integer types are represented in reverse byte order ([**little-endian**](#gt_079478cb-f4c5-4ce5-b72b-2144da5d2ce7)) unless otherwise specified. Each integer takes a whole number of bytes as follows:

**bit:** 1 byte

**tinyint:** 1 byte

**smallint:** 2 bytes

**int:** 4 bytes

**bigint:** 8 bytes

###### Timestamp

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

###### Character and Binary Strings

See [Variable-Length Data Types (section 2.2.5.4.2)](#Section_ce3183a69d8947e8a02fde5a1a1303de) and [Partially Length-Prefixed Data Types (section 2.2.5.4.3)](#Section_7d26a257083e409b81ba897e0c672be0).

###### Fixed-Point Numbers

**smallmoney** is represented as a 4-byte signed integer. The TDS value is the **smallmoney** value multiplied by 104.

**money** is represented as an 8-byte signed integer. The TDS value is the **money** value multiplied by 104. 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.

###### Floating-Point Numbers

**float**(*n*) follows the 32-bit [[IEEE754]](https://go.microsoft.com/fwlink/?LinkId=89903) binary specification when *n* <= 24 and the 64-bit [IEEE754] binary specification when 25 <= *n* <= 53.

###### 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 10s. 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.
  + 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 4, 8, 12, or 16 bytes.

###### GUID

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

###### 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.

##### 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.

1. DB\_NAME = B\_VARCHAR ; database name of the UDT
2. SCHEMA\_NAME = B\_VARCHAR ; schema name of the UDT
3. TYPE\_NAME = B\_VARCHAR ; type name of the UDT
4. MAX\_BYTE\_SIZE = USHORT ; max length in bytes
5. ASSEMBLY\_QUALIFIED\_NAME = US\_VARCHAR ; name of the CLR assembly
6. UDT\_METADATA = ASSEMBLY\_QUALIFIED\_NAME
7. UDT\_INFO\_IN\_COLMETADATA = MAX\_BYTE\_SIZE
8. DB\_NAME
9. SCHEMA\_NAME
10. TYPE\_NAME
11. UDT\_METADATA
12. UDT\_INFO\_IN\_RPC = DB\_NAME ; database name of the UDT
13. SCHEMA\_NAME ; schema name of the UDT
14. TYPE\_NAME ; type name of the UDT
15. UDT\_INFO = UDT\_INFO\_IN\_COLMETADATA ;when sent as part of COLMETADATA
16. /
17. 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]](%5bMS-SSCLRT%5d.pdf#Section_77460aa98c2f4449a65e1d649ebd77fa).

##### XML Values

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

1. SCHEMA\_PRESENT= BYTE;
2. DBNAME = B\_VARCHAR
3. OWNING\_SCHEMA = B\_VARCHAR
4. XML\_SCHEMA\_COLLECTION = US\_VARCHAR
5. XML\_INFO = SCHEMA\_PRESENT
6. [DBNAME OWNING\_SCHEMA
7. 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 that is associated with an XML data type definition stream uses the binary XML structure format, as specified in [[MS-BINXML]](%5bMS-BINXML%5d.pdf#Section_11ab6e8d247244d1a9e6bddf000e12f6).[<14>](#Appendix_A_14" \o "Product behavior note 14)

##### 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 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.

1. VARIANT\_BASETYPE = BYTE ; data type definition
2. VARIANT\_PROPBYTES = BYTE ; see below
3. VARIANT\_PROPERTIES = \*BYTE ; see below
4. VARIANT\_DATAVAL = 1\*BYTE ; actual data value
6. SSVARIANT\_INSTANCE = VARIANT\_BASETYPE
7. VARIANT\_PROPBYTES
8. VARIANT\_PROPERTIES
9. 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** Data types cannot be NULL when inside a sql\_variant. If the value is NULL, the sql\_variant itself has to 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.[<15>](#Appendix_A_15" \o "Product behavior note 15)

##### Table Valued Parameter (TVP) Values

Table Valued Parameters (or User Defined Table Type, as this 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 permitted to be used only as input parameters and do not appear as output parameters or in [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d) columns.

TVPs MUST be sent only by a TDS client that reports itself as a TDS major version 7.3 or later. If a client reporting itself as older than TDS 7.3 attempts to send a TVP, the server MUST reject the request with a TDS protocol error.

###### Metadata

TVPTYPE = %xF3

1. TVP\_TYPE\_INFO = TVPTYPE
2. TVP\_TYPENAME
3. TVP\_COLMETADATA
4. [TVP\_ORDER\_UNIQUE]
5. [TVP\_COLUMN\_ORDERING]
6. TVP\_END\_TOKEN
7. \*TVP\_ROW
8. 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**

1. DBNAME = B\_VARCHAR ; Database where TVP type resides
2. OwningSchema = B\_VARCHAR ; Schema where TVP type resides
3. TypeName = B\_VARCHAR ; TVP type name
4. TVP\_TYPENAME = DbName
5. OwningSchema
6. TypeName

**TVP\_COLMETADATA definition**

1. DbName = B\_VARCHAR ; Database where TVP type resides
2. fNullable = BIT ; Column is nullable - %x01
3. fCaseSen = BIT ; Column is case-sensitive - %x02
4. usUpdateable = 2BIT ; 2-bit value, one of:
5. ; 0 = ReadOnly - %x00
6. ; 1 = ReadWrite - %x04
7. ; 2 = Unknown - %x08
8. fIdentity = BIT ; Column is identity column - %x10
9. fComputed = BIT ; Column is computed - %x20
10. usReservedODBC = 2BIT ; Reserved bits for ODBC - %x40+80
11. fFixedLenCLRType = BIT ; Fixed length CLR type - %x100
12. fDefault = BIT ; Column is default value - %x200
13. usReserved = 6BIT ; Six leftover reserved bits
14. Flags = fNullable
15. fCaseSen
16. usUpdateable
17. fIdentity
18. fComputed
19. usReservedODBC
20. fFixedLenCLRType
21. fDefault
22. usReserved
23. Count = USHORT ; Column count up to 1024 max
24. ColName = B\_VARCHAR ; Name of column
25. UserType = ULONG ; UserType of column
26. TypeColumnMetaData = UserType
27. Flags
28. TYPE\_INFO
29. ColName ; Column metadata instance
30. TVP\_NULL\_TOKEN = %xFFFF
31. TVP\_COLMETADATA = TVP\_NULL\_TOKEN / ( Count (<Count> TvpColumnMetaData) )

DbName, OwningSchema, and TypeName are limited to 128 [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) 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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) or function where parameter metadata is available on the server side, the client can send all zero-length strings for TVP\_TYPENAME.
* If the TVP is a parameter to an ad-hoc [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d), 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**

| 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 |

###### Optional Metadata Tokens

**TVP\_ORDER\_UNIQUE definition**

1. TVP\_ORDER\_UNIQUE\_TOKEN = %x10
2. Count = USHORT ; Count of ColNums to follow
3. ColNum = USHORT ; A single-column ordinal
4. fOrderAsc = BIT ; Column-ordered ascending – %x01
5. fOrderDesc = BIT ; Column-ordered descending – %x02
6. fUnique = BIT ; Column is in unique set – %x04
7. Reserved1 = 5BIT ; Five reserved bits
8. OrderUniqueFlags = fOrderAsc
9. fOrderDesc
10. fUnique
11. Reserved1
12. TVP\_ORDER\_UNIQUE = TVP\_ORDER\_UNIQUE\_TOKEN
13. ( 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.

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.

1. TVP\_COLUMN\_ORDERING\_TOKEN = %x11
2. Count = USHORT ; Count of ColNums to follow
3. ColNum = USHORT ; A single-column ordinal
4. TVP\_COLUMN\_ORDERING = TVP\_COLUMN\_ORDERING\_TOKEN
5. ( 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, say, a TVP is defined as the following:

1. TVP\_COLUMN\_ORDERING = create type myTvpe as table (f1 int / f2 varchar (max) / f3 datetime)

Then, 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.

In this case, the TVP\_COLUMN\_ORDERING token on the wire for this example would be:

1. 11 ; TVP\_COLUMN\_ORDERING\_TOKEN
2. 03 00 ; Count - Number of ColNums to follow.
3. 01 00 ; ColNum - TVP column ordinal 1 is sent first in TVP\_COLMETADATA.
4. 03 00 ; ColNum - TVP column ordinal 3 is sent second in TVP\_COLMETADATA.
5. 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**

1. TVP\_ROW\_TOKEN = %x01 ; A row as defined by TVP\_COLMETADATA follows
2. TvpColumnData = TYPE\_VARBYTE ; Actual value must match metadata for the column
3. AllColumnData = \*TvpColumnData ; Chunks of data, one per non-default column defined
4. ; in TVP\_COLMETADATA
5. TVP\_ROW = TVP\_ROW\_TOKEN
6. AllColumnData
7. TVP\_END\_TOKEN = %x00 ; Terminator tag for TVP type, meaning
8. ; no more TVP\_ROWs to follow and end of
9. ; 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.

###### TDS Type Restrictions

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

| TDS type | Replacement type |
| --- | --- |
| Binary | BigBinary |
| VarBinary | BigVarBinary |
| 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 are not to be confused with data types for a database server that supports SQL.

#### 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](#Section_ce3183a69d8947e8a02fde5a1a1303de).

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.

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.

1. USHORTMAXLEN = %xFFFF
2. TYPE\_INFO = FIXEDLENTYPE
3. /
4. (VARLENTYPE TYPE\_VARLEN [COLLATION])
5. /
6. (VARLENTYPE TYPE\_VARLEN [PRECISION SCALE])
7. /
8. (VARLENTYPE SCALE) ; (introduced in TDS 7.3)
9. /
10. VARLENTYPE ; (introduced in TDS 7.3)
11. /
12. (PARTLENTYPE
13. [USHORTMAXLEN]
14. [COLLATION]
15. [XML\_INFO]
16. [UDT\_INFO])

#### Encryption Key Rule Definition

The EK\_INFO rule applies to messages that have encrypted values and describes the encryption key information. The encryption key information includes the various encryption key values that are obtained by securing an encryption key by using different master keys. This rule applies only if the column encryption feature is negotiated by the client and the server and is turned ON.

1. Count = BYTE
2. EncryptedKey = US\_VARBYTES
3. KeyStoreName = B\_VARCHAR
4. KeyPath = US\_VARCHAR
5. AsymmetricAlgo = B\_VARCHAR
6. EncryptionKeyValue = EncryptedKey
7. KeyStoreName
8. KeyPath
9. AsymmetricAlgo
10. DatabaseId = ULONG
11. CekId = ULONG
12. CekVersion = ULONG
13. CekMDVersion = ULONGLONG
14. EK\_INFO = DatabaseId
15. CekId
16. CekVersion
17. CekMDVersion
18. Count
19. \*EncryptionKeyValue

| Parameter | Description |
| --- | --- |
| Count | The count of EncryptionKeyValue elements that are present in the message. |
| EncryptedKey | The ciphertext containing the encryption key that is secured with the master. |
| KeyStoreName | The key store name component of the location where the master key is saved. |
| KeyPath | The key path component of the location where the master key is saved. |
| AsymmetricAlgo | The name of the algorithm that is used for encrypting the encryption key. |
| EncryptionKeyValue | The metadata and encrypted value that describe an encryption key. This is enough information to allow retrieval of plaintext encryption keys. |
| DatabaseId | A 4 byte integer value that represents the database ID where the column encryption key is stored. |
| CekId | An identifier for the column encryption key. |
| CekVersion | The key version of the column encryption key. |
| CekMDVersion | The metadata version for the column encryption key. |

#### Data Packet 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](#Section_c060af9c6db74360954cc28a132c8949).

1. ALTMETADATA\_TOKEN = %x88
2. ALTROW\_TOKEN = %xD3
3. COLMETADATA\_TOKEN = %x81
4. COLINFO\_TOKEN = %xA5
5. DONE\_TOKEN = %xFD
6. DONEPROC\_TOKEN = %xFE
7. DONEINPROC\_TOKEN = %xFF
8. ENVCHANGE\_TOKEN = %xE3
9. ERROR\_TOKEN = %xAA
10. FEATUREEXTACK\_TOKEN = %xAE ; (introduced in TDS 7.4)
11. FEDAUTHINFO\_TOKEN = %xEE ; (introduced in TDS 7.4)
12. INFO\_TOKEN = %xAB
13. LOGINACK\_TOKEN = %xAD
14. NBCROW\_TOKEN = %xD2 ; (introduced in TDS 7.3)
15. OFFSET\_TOKEN = %x78
16. ORDER\_TOKEN = %xA9
17. RETURNSTATUS\_TOKEN = %x79
18. RETURNVALUE\_TOKEN = %xAC
19. ROW\_TOKEN = %xD1
20. SESSIONSTATE\_TOKEN = %xE4 ; (introduced in TDS 7.4)
21. SSPI\_TOKEN = %xED
22. TABNAME\_TOKEN = %xA4

### Packet Header Message Type Stream Definition

#### Bulk Load BCP

**Stream Name:**

1. 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]](https://go.microsoft.com/fwlink/?LinkId=154273).
* A sample BulkLoadBCP message is in section [4.12](#Section_43dac55aa31a4122ac0620cfedf76811).

**Stream-Specific Rules:**

1. BulkLoad\_METADATA = COLMETADATA\_TOKEN
2. BulkLoad\_ROW = ROW\_TOKEN
3. BulkLoad\_DONE = DONE\_TOKEN

**Submessage Definition:**

1. BulkLoadBCP = BulkLoad\_METADATA
2. \*BulkLoad\_ROW
3. 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.

#### Bulk Load Update Text/Write Text

**Stream Name:**

1. 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 **"UPDATETEXT BULK"** or **"WRITETEXT BULK"** T-SQL statement to the server. For information about the UPDATETEXT BULK and WRITETEXT BULK syntax, see [[MSDN-UPDATETEXT]](https://go.microsoft.com/fwlink/?LinkId=154272) and [[MSDN-WRITETEXT]](https://go.microsoft.com/fwlink/?LinkId=154269), respectively.
* The server returns a RETURNVALUE token containing the new timestamp for this column.

**Stream-Specific Rules:**

1. BulkData = L\_VARBYTE

**Sub Message Definition:**

1. BulkLoadUTWT = BulkData

**Stream Parameter Details**

| Parameter | Description |
| --- | --- |
| BulkData | Contains the BulkData length and BulkData data within the L\_VARBYTE. |

#### Federated Authentication Token

**Stream Name:**

1. FEDAUTH

**Stream Function:**

An authentication token for [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19).[<16>](#Appendix_A_16" \o "Product behavior note 16)

**Stream Comments:**

* Packet header type 0x08.
* This stream contains the client’s federated authentication token, generated by a client library that is supported by the server, and any other information, as laid out in the rules for the particular bFedAuthLibrary that is indicated in the FEDAUTH FeatureExt in the Login message.
* The server MUST respond with a Login Response message or an error.

**Stream-Specific Rules:**

1. DataLen = DWORD
2. FedAuthToken = L\_VARBYTE
3. Nonce = 32BYTE

**Stream Definition:**

Stream is defined based on the bFedAuthLibrary that is used in Login message FEDAUTH FeatureExt. This message MUST not be sent for any values of bFedAuthLibrary that are not listed in this section.

When bFedAuthLibrary is Active Directory Authentication Library (ADAL) [that is, 0x02]:

1. FEDAUTH = DataLen
2. FedAuthToken
3. [Nonce]

**Stream Parameter Details**

| Parameter | Description |
| --- | --- |
| DataLen | The total length of the data in the Federated Authentication Token message that follows this field. DataLen does not include the size that is used for the DataLen field itself. |
| FedAuthToken | Contains the federated authentication token data that is generated by the federated authentication library. The federated authentication library that is used to generate the token MUST be the same library that is specified as bFedAuthLibrary in the client’s Login FEDAUTH FeatureExt message. |
| Nonce | The nonce, if provided by the server during the pre-login exchange, that is echoed back to the server by the client. If the server provided a nonce in the pre-login exchange, the client MUST echo the nonce back to the server in this field. If the server did not provide a nonce to the client in the pre-login exchange, this field MUST NOT be included in the stream. |

#### LOGIN7

**Stream Name:**

1. 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 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](#Section_ce5ad23f6bf84fa594266b0d36e14da2).

**Stream-Specific Rules:**

1. Length = DWORD
2. TDSVersion = DWORD
3. PacketSize = DWORD
4. ClientProgVer = DWORD
5. ClientPID = DWORD
6. ConnectionID = DWORD
7. fByteorder = BIT
8. fChar = BIT
9. fFloat = 2BIT
10. fDumpLoad = BIT
11. fUseDB = BIT
12. fDatabase = BIT
13. fSetLang = BIT
14. OptionFlags1 = fByteorder
15. fChar
16. fFloat
17. fDumpLoad
18. fUseDB
19. fDatabase
20. fSetLang
21. fLanguage = BIT
22. fODBC = BIT
23. fTranBoundary = BIT ; (removed in TDS 7.2)
24. fCacheConnect = BIT ; (removed in TDS 7.2)
25. fUserType = 3BIT
26. fIntSecurity = BIT
27. OptionFlags2 = fLanguage
28. fODBC
29. (fTransBoundary / FRESERVEDBIT)
30. (fCacheConnect / FRESERVEDBIT)
31. fUserType
32. fIntSecurity
33. fSQLType = 4BIT
34. fOLEDB = BIT ; (introduced in TDS 7.2)
35. fReadOnlyIntent = BIT ; (introduced in TDS 7.4)
36. TypeFlags = fSQLType
37. (FRESERVEDBIT / fOLEDB)
38. (FRESERVEDBIT / fReadOnlyIntent)
39. 2FRESERVEDBIT
40. fChangePassword = BIT ; (introduced in TDS 7.2)
41. fUserInstance = BIT ; (introduced in TDS 7.2)
42. fSendYukonBinaryXML = BIT ; (introduced in TDS 7.2)
43. fUnknownCollationHandling = BIT ; (introduced in TDS 7.3)
44. fExtension = BIT ; (introduced in TDS 7.4)
45. OptionFlags3 = (FRESERVEDBIT / fChangePassword)
46. (FRESERVEDBIT / fSendYukonBinaryXML)
47. (FRESERVEDBIT / fUserInstance)
48. (FRESERVEDBIT / fUnknownCollationHandling)
49. (FRESERVEDBIT / fExtension)
50. 3FRESERVEDBIT
51. ClientTimZone = LONG;
52. ClientLCID = LCID
53. ColFlags
54. Version
55. ibHostName = USHORT
56. cchHostName = USHORT
57. ibUserName = USHORT
58. cchUserName = USHORT
59. ibPassword = USHORT
60. cchPassword = USHORT
61. ibAppName = USHORT
62. cchAppName = USHORT
63. ibServerName = USHORT
64. cchServerName = USHORT
65. ibUnused = USHORT
66. cbUnused = USHORT
67. ibExtension = USHORT ; (introduced in TDS 7.4)
68. cbExtension = USHORT ; (introduced in TDS 7.4)
69. ibCltIntName = USHORT
70. cchCltIntName = USHORT
71. ibLanguage = USHORT
72. cchLanguage = USHORT
73. ibDatabase = USHORT
74. cchDatabase = USHORT
75. ClientID = 6BYTE
76. ibSSPI = USHORT
77. cbSSPI = USHORT
78. ibAtchDBFile = USHORT
79. cchAtchDBFile = USHORT
80. ibChangePassword = USHORT ; (introduced in TDS 7.2)
81. cchChangePassword = USHORT ; (introduced in TDS 7.2)
82. cbSSPILong = DWORD ; (introduced in TDS 7.2)
83. OffsetLength = ibHostName
84. cchHostName
85. ibUserName
86. cchUserName
87. ibPassword
88. cchPassword
89. ibAppName
90. cchAppName
91. ibServerName
92. cchServerName
93. (ibUnused / ibExtension)
94. (cchUnused / cbExtension)
95. ibCltIntName
96. cchCltIntName
97. ibLanguage
98. cchLanguage
99. ibDatabase
100. cchDatabase
101. ClientID
102. ibSSPI
103. cbSSPI
104. ibAtchDBFile
105. cchAtchDBFile
106. ibChangePassword
107. cchChangePassword
108. cbSSPILong

**Note**  The ClientLCID value is no longer used to set language parameters and is ignored.

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.

1. Data = \*BYTE
2. FeatureId = BYTE ; (introduced in TDS 7.4)
3. FeatureDataLen = DWORD ; (introduced in TDS 7.4)
4. FeatureData = \*BYTE ; (introduced in TDS 7.4)
5. TERMINATOR = %xFF ; signal of end of feature option
6. FeatureOpt = (FeatureId
7. FeatureDataLen
8. FeatureData)
9. /
10. TERMINATOR
11. FeatureExt = 1\*FeatureOpt ; (introduced in TDS 7.4)

**Stream Definition:**

1. LOGIN7 = Length
2. TDSVersion
3. PacketSize
4. ClientProgVer
5. ClientPID
6. ConnectionID
7. OptionFlags1
8. OptionFlags2
9. TypeFlags
10. (FRESERVEDBYTE / OptionFlags3)
11. ClientTimZone
12. ClientLCID
13. OffsetLength
14. Data
15. [FeatureExt]

**Stream Parameter Details**

| 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.[<17>](#Appendix_A_17" \o "Product behavior note 17)  For information about what the server sends to the client, see the [LOGINACK](#Section_490e563dcc6e4c86bb95ef0186b98032) token. |
| PacketSize | The packet size being requested by the client. |
| ClientProgVer | The version of the [**interface**](#gt_95913fbd-3262-47ae-b5eb-18e6806824b9) 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](#Section_bbc22f15e1a04338a169c79819d39b1c). * fByteOrder: The byte order used by client for numeric and datetime data types.   + 0 = ORDER\_X86   + 1 = ORDER\_68000[<18>](#Appendix_A_18" \o "Product behavior note 18) * fChar: The character set used on the client.   + 0 = CHARSET\_ASCII   + 1 = CHARSET\_EBCDIC * fFloat: The type of floating point representation used by the client.[<19>](#Appendix_A_19" \o "Product behavior note 19)   + 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 requires 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 * 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 requires 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.[<20>](#Appendix_A_20" \o "Product behavior note 20)   + 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   + 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.[<21>](#Appendix_A_21" \o "Product behavior note 21)   + 0 = OLEDB\_OFF   + 1 = OLEDB\_ON * fReadOnlyIntent: This bit was 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.[<22>](#Appendix_A_22" \o "Product behavior note 22) * 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]](https://go.microsoft.com/fwlink/?LinkId=119987) and [[MS-LCID]](%5bMS-LCID%5d.pdf#Section_70feba9f294e491eb6eb56532684c37f) 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. |
| 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:   1. ibFeatureExtLong = DWORD   2. 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 by using the NIC address). ClientID is the MAC address of the physical network layer. It is used to identify the client that is connecting to the server. This value is mainly informational, and no processing steps on the server side use it. * 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 TDS 7.2. |
| Data | The actual variable-length data portion referred to by OffsetLength. |
| FeatureId | The unique identifier number of a feature. The available features are described in the following table.  Introduced in TDS 7.4. |
| FeatureDataLen | The length, in bytes, of FeatureData for the corresponding FeatureId.  Introduced in TDS 7.4. |
| FeatureData | Data of the feature. Each feature defines its own data format.  The data for existing features are defined in the following table.  Introduced in TDS 7.4. |
| 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 whether the server accepts the feature option.  Optionally, a feature can use a [FEATUREEXTACK](#Section_2eb82f8e11f046dcb42d27302fa4701a) token to acknowledge the feature along with LOGINACK. The detailed acknowledge data SHOULD be defined by the feature itself.  Introduced in TDS 7.4. |

The following table defines the options that are available in FeatureExt.

| FeatureId | FeatureData Description |
| --- | --- |
| %0x01  (SESSIONRECOVERY)  (introduced in TDS 7.4) | Session Recovery feature. This feature is used to recover the session state of a previous connection. Content is defined as follows:   1. Length = DWORD 2. RecoveryDatabase = B\_VARCHAR 3. RecoveryCollation = BYTELEN [COLLATION] 4. RecoveryLanguage = B\_VARCHAR 5. SessionRecoveryData = Length 6. RecoveryDatabase 7. RecoveryCollation 8. RecoveryLanguage 9. SessionStateDataSet 10. InitSessionRecoveryData = SessionRecoveryData 11. SessionRecoveryDataToBe = SessionRecoveryData 12. FeatureData = [InitSessionRecoveryData SessionRecoveryDataToBe]   The Length field is the length, in bytes, of SessionRecoveryData excluding the Length field itself. SessionStateDataSet is described in section [2.2.7.20](#Section_626fbe19f3564599ba17c70f44005106). 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**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) characters.  There are two sets of SessionRecoveryData. The data for the first set, InitSessionRecoveryData, SHOULD come from the initial login response data of 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 is 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.   After 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 and 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 client and 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 that were received from the server. The client MUST NOT try to recover a dead connection if the any latest fRecovery bit is FALSE. |
| %0x02  (FEDAUTH)[<23>](#Appendix_A_23" \o "Product behavior note 23)  (introduced in TDS 7.4) | The presence of the FEDAUTH FeatureExt indicates that the client is authenticating by [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19). If the FEDAUTH FeatureId is present, the value of fIntSecurity MUST be 0. The format of the data is as described below based on the bFedAuthLibrary that is used.  bFedAuthLibrary = 0x7F is a reserved value.  When the bFedAuthLibrary is Live ID Compact Token, the format is as follows:   1. bFedAuthLibrary = 7BIT 2. fFedAuthEcho = BIT 3. Options = bFedAuthLibrary 4. fFedAuthEcho 5. FedAuthToken = L\_VARBYTE 6. Nonce = 32BYTE 7. ChannelBindingToken = BYTESTREAM 8. Signature = 32BYTE 9. SignedData = Nonce 10. [ChannelBindingToken] 11. Signature 12. FeatureData = Options 13. FedAuthToken 14. 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 the response was not tampered with. 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 pre-login 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]](https://go.microsoft.com/fwlink/?LinkId=328921) 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 bFedAuthLibrary is Security Token, the format is as follows:   1. bFedAuthLibrary = 7BIT 2. fFedAuthEcho = BIT 3. Options = bFedAuthLibrary 4. fFedAuthEcho 5. FedAuthToken = L\_VARBYTE 6. Nonce = 32BYTE 7. OtherData = Nonce 8. FeatureData = Options 9. FedAuthToken 10. [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 the response was not tampered with. 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 and echoed back to the server by the client. This field MUST be present if the server’s PRELOGIN message included a NONCE field. Otherwise, this field MUST NOT be present.  When bFedAuthLibrary is Active Directory Authentication Library (ADAL) [that is, 0x02], the format is as follows:   1. bFedAuthLibrary = 7BIT 2. fFedAuthEcho = BIT 3. Workflow = BYTE 4. Options = bFedAuthLibrary 5. fFedAuthEcho 6. Workflow   bFedAuthLibrary: 7 bits, collectively treated as a 7-bit unsigned integer that indicates the library that is used by the client for federated authentication.  0x02 = ADAL. After the client establishes the intent to use ADAL, for which additional information is required by the client to generate a token, the server MUST respond with the Federated Authentication Information token to the client with "FedAuthInfoIDs: STSURL, SPN".  fFedAuthEcho: The intention of this flag is for the client to echo the server’s FEDAUTHREQUIRED pre-login option so that the server can validate that the response was not tampered with. The client MUST assign this flag to 1 if and only if the server’s PRELOGIN Response contains a FEDAUTHREQUIRED option with a B\_FEDAUTHREQUIRED value of 0x01.  Workflow: Indicates the ADAL (that is, 0x02) workflow that is being used.  0x01 = Username/password. A username and password are passed to ADAL to retrieve a token.  0x02 = Integrated. A Windows identity is passed to ADAL to retrieve a token.  All other values of bFedAuthLibrary are reserved. |
| %0x04  (COLUMNENCRYPTION)  (introduced in TDS 7.4) | The presence of the COLUMNENCRYPTION FeatureExt indicates that the client SHOULD[<24>](#Appendix_A_24" \o "Product behavior note 24) be capable of performing cryptographic operations on data. The feature data are described as follows:   1. Length = DWORD 2. COLUMNENCRYPTION\_VERSION = BYTE 3. FeatureData = Length 4. COLUMNENCRYPTION\_VERSION 5. \*EnclaveType   COLUMNENCRYPTION\_VERSION: This field describes the cryptographic protocol version that the client understands. The values of this field are as follows:   * 1 = The client supports column encryption without [**enclave computations**](#gt_6fe5534f-5cd8-4ab6-aba4-637a4344eda0). * 2 = The client MAY[<25>](#Appendix_A_25" \o "Product behavior note 25) support column encryption when encrypted data require enclave computations.   EnclaveType: This field is a string that MAY[<26>](#Appendix_A_26" \o "Product behavior note 26) be populated by the server and used by the client to identify the type of [**enclave**](#gt_ef41e9e0-3e5d-432f-96d6-39515bdc5340) that the server is configured to use. During login for the initial connection, the client can request COLUMNENCRYPTION with the Length as 1, and either 1 or 2 as the value of COLUMNENCRYPTION\_VERSION, and no **EnclaveType**. When the client sends COLUMNENCRYPTION\_VERSION as 2 and no **EnclaveType**, the server MUST return COLUMNENCRYPTION\_VERSION as 2 together with the value of **EnclaveType**, if the server supports this column encryption version. If **EnclaveType** is not returned and the column encryption version is returned as 2, the client driver MUST raise an error. |
| %0x05  (GLOBALTRANSACTIONS)[<27>](#Appendix_A_27" \o "Product behavior note 27)  (introduced in TDS 7.4) | The presence of the GLOBALTRANSACTIONS FeatureExt indicates that the client is capable of performing [**Global Transactions**](#gt_57552b13-b14e-4601-9621-500ce3297d15). The feature data is described as follows:   1. Length = DWORD ; Value is (0) 2. FeatureData = NO DATA   NO DATA: No feature data is sent with the GLOBALTRANSACTIONS FeatureExt. |
| %0x08  (AZURESQLSUPPORT)  (introduced in TDS 7.4) | The presence of the AZURESQLSUPPORT FeatureExt indicates whether the client MAY[<28>](#Appendix_A_28" \o "Product behavior note 28) support failover partner login with read-only intent in Azure SQL Database. For information about failover partner, see [[MSDOCS-DBMirror]](https://go.microsoft.com/fwlink/?linkid=874052).  The feature data is described as follows:   1. Length = DWORD 2. FeatureData = BYTE   BYTE: The Bit 0 flag specifies whether failover partner login with read-only intent is supported. The values of this BYTE are as follows:   * 0 = The server does not support the AZURESQLSUPPORT feature extension. * 1 = The server supports the AZURESQLSUPPORT feature extension. |
| %xFF  (TERMINATOR) | This option signals the end of the FeatureExt feature and MUST be the feature's last option. |

**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.

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 or ibChangePassword, 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 or ibChangePassword, the server SHOULD first do a bit-XOR with 0xA5 (10100101) and then swap the four high bits with the four low bits.

#### PRELOGIN

**Stream Name:**

1. 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](#Section_9420b4a3eb9f4f5e90bd3160444aa5a7).

**Stream-Specific Rules:**

1. UL\_VERSION = ULONG ; version of the sender
2. US\_SUBBUILD = USHORT ; sub-build number of the sender
3. B\_FENCRYPTION = BYTE
4. B\_INSTVALIDITY = \*BYTE %x00 ; name of the instance
5. ; of the database server that supports SQL
6. ; or just %x00
7. UL\_THREADID = ULONG ; client application thread id
8. ; used for debugging purposes
9. B\_MARS = BYTE ; sender requests MARS support
10. GUID\_CONNID = 16BYTE ; client application trace id
11. ; used for debugging purposes
12. GUID\_ActivityID = 16bytes ; client application activity id
13. ; used for debugging purposes
14. ActivitySequence = ULONG ; client application activity sequence
15. ; used for debugging purposes
16. B\_FEDAUTHREQUIRED = BYTE ; authentication library requirement of the sender
17. ; when using Integrated Authentication identity
18. ; introduced in TDS 7.4
19. NONCE = 32BYTE ; nonce to be encrypted by using session key from
20. ; federated authentication handshake
21. TERMINATOR = %xFF ; signals end of PRELOGIN message
22. PL\_OPTION\_DATA = \*BYTE ; actual data for the option
23. PL\_OFFSET = USHORT ; big endian
24. PL\_OPTION\_LENGTH = USHORT ; big endian
25. PL\_OPTION\_TOKEN = BYTE ; token value representing the option
26. PRELOGIN\_OPTION = (PL\_OPTION\_TOKEN
27. PL\_OFFSET
28. PL\_OPTION\_LENGTH)
29. /
30. TERMINATOR
31. SSL\_PAYLOAD = \*BYTE ; SSL handshake raw payload

**Stream Definition:**

1. PRELOGIN = (\*PRELOGIN\_OPTION
2. \*PL\_OPTION\_DATA)
3. /
4. 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 composed of major version (1 byte), minor version (1 byte), and build number (2 bytes). It is represented in network byte order ([**big-endian**](#gt_6f6f9e8e-5966-4727-8527-7e02fb864e7e)).  On x86 platforms, UL\_VERSION is prepared as follows:  US\_BUILD = SwapBytes (VER\_SQL\_BUILD);  UL\_VERSION = ( (US\_BUILD <16>) / (VER\_SQL\_MINOR <8>) / ( VER\_SQL\_MAJOR) )  SwapBytes is used to swap bytes. For example, SwapBytes(0x106A)= 0x6A10. |
| 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 |
| TRACEID | 0x05 | 1. PL\_OPTION\_DATA = GUID\_CONNID 2. ACTIVITY\_GUID 3. SEQUENCE\_ID |
| FEDAUTHREQUIRED[<29>](#Appendix_A_29" \o "Product behavior note 29) | 0x06 | PL\_OPTION\_DATA = B\_FEDAUTHREQUIRED  Introduced in TDS 7.4. |
| NONCEOPT | 0x07 | 1. 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](#Section_8bfcbcd21baf47928fff86a63a9e90fa).
* 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 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 responds 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 |
| 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 requires 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 client and server negotiate to enable encryption, an SSL handshake takes place immediately after the initial PRELOGIN/table response message exchange. The SSL payloads MUST be transported as data in TDS packets with the message type set to 0x12 in the packet header. For example:

1. 0x 12 01 00 4e 00 00 00 00// Packet Header
2. 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**](#gt_71dd1dd2-c167-49a8-a5f5-6b0df5c8b48a) 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[<30>](#Appendix_A_30" \o "Product behavior note 30) 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 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.

#### RPC Request

**Stream Name:**

1. 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**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d).
* A sample RPCRequest message is in section [4.8](#Section_1469b2fa6cab42e991f9044d358b306b).

**Stream-Specific Rules:**

1. ProcID = USHORT
2. ProcIDSwitch = %xFF %xFF
3. ProcName = US\_VARCHAR
4. NameLenProcID = ProcName
5. /
6. (ProcIDSwitch ProcID)
7. fWithRecomp = BIT
8. fNoMetaData = BIT
9. fReuseMetaData = BIT
10. OptionFlags = fWithRecomp
11. fNoMetaData
12. fReuseMetaData
13. 13FRESERVEDBIT
14. fByRefValue = BIT
15. fDefaultValue = BIT
16. fEncrypted = BIT
17. StatusFlags = fByRefValue
18. fDefaultValue
19. 1FRESERVEDBIT
20. fEncrypted
21. 4FRESERVEDBIT
22. ParamMetaData = B\_VARCHAR
23. StatusFlags
24. (TYPE\_INFO / TVP\_TYPE\_INFO) ; (TVP\_TYPE\_INFO introduced in TDS 7.3)
25. ParamLenData = TYPE\_VARBYTE
26. EncryptionAlgo = BYTE ; (introduced in TDS 7.4)
27. AlgoName = B\_VARCHAR ; (introduced in TDS 7.4)
28. EncryptionType = BYTE ; (introduced in TDS 7.4)
29. NormVersion = BYTE ; (introduced in TDS 7.4)
30. DatabaseId = ULONG ; (introduced in TDS 7.4)
31. CekId = ULONG ; (introduced in TDS 7.4)
32. CekVersion = ULONG ; (introduced in TDS 7.4)
33. CekMDVersion = ULONGLONG ; (introduced in TDS 7.4)
34. ParamCipherInfo = TYPE\_INFO
35. EncryptionAlgo
36. [AlgoName]
37. EncryptionType
38. DatabaseId
39. CekId
40. CekVersion
41. CekMDVersion
42. NormVersion
43. ParameterData = ParamMetaData
44. ParamLenData
45. [ParamCipherInfo]
46. EnclavePackage = L\_VARBYTE ; (introduced in TDS 7.4)
47. BatchFlag = %x80 / %xFF ; (changed to %xFF in TDS 7.2)
48. NoExecFlag = %xFE ; (introduced in TDS 7.2)
49. RPCReqBatch = NameLenProcID
50. OptionFlags
51. \*EnclavePackage
52. \*ParameterData

The length for the instance value of UDTs is specified as a ULONGLONG. Also, 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:**

1. RPCRequest = ALL\_HEADERS
2. RPCReqBatch
3. \*((BatchFlag / NoExecFlag) RPCReqBatch)
4. [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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) 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 * 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](#Section_bbc22f15e1a04338a169c79819d39b1c):   * fWithRecomp: 1 if RPC is sent with the "with recompile" option. * fNoMetaData: The server sends NoMetaData only if fNoMetadata is set to 1 in the request (see COLMETADATA, section [2.2.7.4](#Section_58880b9f381c43b2bf8b0727a98c4f4c)).[<31>](#Appendix_A_31" \o "Product behavior note 31) * 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. * fEncrypted: 1 if the parameter that is being passed is encrypted. This flag is valid only when the column encryption feature is negotiated by client and server and is turned on. |
| 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). |
| EncryptionAlgo | This byte describes the encryption algorithm that is used. For a custom encryption algorithm, the EncryptionAlgo value MUST be set to 0 and the actual encryption algorithm MUST be inferred from the AlgoName. For all other values, AlgoName MUST NOT be sent.  If the value is set to 1, the encryption algorithm that is used is AEAD\_AES\_256\_CBC\_HMAC\_SHA512, as described in [[IETF-AuthEncr]](https://go.microsoft.com/fwlink/?LinkId=524322) section 5.4. |
| AlgoName | Algorithm name literal that is used for encrypting the plaintext value. This is an optional field and MUST be sent when EncryptionAlgo = 0. For all other values of EncryptionAlgo, this field MUST NOT be sent. |
| EncryptionType | This byte describes the flavor of encryption algorithm that is used. The values of this field are as follows:  1 = deterministic encryption.  2 = randomized encryption. |
| NormVersion | Reserved for future use. The value MUST be set to 1. |
| DatabaseId | A 4-byte integer value that represents the database ID where the column encryption key is stored. |
| CekId | An identifier for the column encryption key. |
| CekVersion | The key version of the column encryption key. |
| CekMDVersion | The metadata version for the column encryption key. |
| ParamCipherInfo | The description of the parameter encryption information when the parameter is transparently encrypted. It defines the original TYPE\_INFO of the data that is encrypted, the encryption algorithm that is used, the normalization version, the id of the database containing the column encryption key used for encryption, the id of the column encryption key, the version of the column encryption key, and the version of the column encryption key metadata. These fields MUST be sent only when fEncrypted is set to 1. |
| EnclavePackage | An encrypted byte package that MAY[<32>](#Appendix_A_32" \o "Product behavior note 32) be generated by the client. This package contains information that is required by the server-side [**enclave**](#gt_ef41e9e0-3e5d-432f-96d6-39515bdc5340) to perform computations on encrypted columns. The package has an internal structure that is irrelevant to the TDS protocol between client and server. The server forwards the byte array to the enclave without interpreting it, and the enclave decodes the byte array.  Introduced in TDS 7.4. |
| 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 RPC, just return an error message, followed by DONEPROC, and then execute the next RPC. |

#### SQLBatch

**Stream Name:**

1. 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](#Section_b05b006b3cbf404bbcaf7ec584b54706).

**Stream-Specific Rules:**

1. SQLText = UNICODESTREAM

**Stream Definition:**

1. SQLBatch = ALL\_HEADERS
2. \*EnclavePackage ; (described in section 2.2.6.6)
3. SQLText

The [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) stream contains the text of the batch. The following is an example of a valid value for SQLText.

1. Select author\_id from Authors

#### SSPI Message

**Stream Name:**

1. SSPI

**Stream Function:**

A request to supply data for [**Security Support Provider Interface (SSPI)**](#gt_fb216516-748b-4873-8bdd-64c5f4da9920) security. Note that SSPI uses the [**Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)**](#gt_bc2f6b5e-e5c0-408b-8f55-0350c24b9838) [[RFC4178]](https://go.microsoft.com/fwlink/?LinkId=90461) 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**](#gt_95913fbd-3262-47ae-b5eb-18e6806824b9) with the server's response.
* 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](#Section_dc57840ad13b43a1aad35425afadbc0c).

**Stream-Specific Rules:**

1. SSPIData = BYTESTREAM

**Stream Definition:**

1. SSPI = SSPIData

**Stream Parameter Details**

| Parameter | Description |
| --- | --- |
| SSPIData | The SSPIData length and SSPIData data using US\_VARCHAR format. |

#### Transaction Manager Request

**Stream Name:**

1. 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)**](#gt_177a71e6-7cf0-48c1-b169-063da349648a) implemented to the DTC Interface Specification. For more information about DTC, see [[MSDN-DTC]](https://go.microsoft.com/fwlink/?LinkId=89994).

**Stream Comments:**

* Packet header type 0x0E.
* A sample Transaction Manager Request message is given in section [4.13](#Section_9f10990a9c744a3989da2a3b05effacb).

**Stream-Specific Rules:**

1. RequestType = USHORT

**Stream Definition:**

1. TransMgrReq = ALL\_Headers
2. RequestType
3. [RequestPayload]

RequestPayload details are as specified in the following table.

**Stream Parameter Details**

| Parameter | Description |
| --- | --- |
| RequestType | The types of [**transaction manager**](#gt_4553803e-9d8d-407c-ad7d-9e65e01d6eb3) operations that are requested 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d) 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.   Request types 5 through 9 were introduced in TDS 7.2. |
| RequestPayload | * For RequestType TM\_GET\_DTC\_ADDRESS: The RequestPayload SHOULD be a zero-length US\_VARBYTE.  1. 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]](https://go.microsoft.com/fwlink/?LinkId=146594)).  1. RequestPayload = US\_VARBYTE  * For RequestType TM\_BEGIN\_XACT:  1. ISOLATION\_LEVEL = BYTE 2. BEGIN\_XACT\_NAME = B\_VARBYTE 3. RequestPayload = ISOLATION\_LEVEL 4. 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:  1. fBeginXact = BIT 2. XACT\_FLAGS = fBeginXact 3. 7FRESERVEDBIT 4. ISOLATION\_LEVEL = BYTE 5. XACT\_NAME = B\_VARBYTE 6. BEGIN\_XACT\_NAME = B\_VARBYTE 7. RequestPayload = XACT\_NAME 8. XACT\_FLAGS 9. [ISOLATION\_LEVEL 10. 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](#Section_bbc22f15e1a04338a169c79819d39b1c).  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 for the session, once the transaction ends.  If fBeginXact is 0, BEGIN\_XACT\_NAME SHOULD NOT be present. If fBeginXact is 1, BEGIN\_XACT\_NAME can be nonempty.  If fBeginXact is 1, a new transaction MUST be started. If BEGIN\_XACT\_NAME is nonempty, the new transaction MUST be given the specified name.  See the definition for isolation level at the end of this table.   * For RequestType TM\_ROLLBACK\_XACT:  1. fBeginXact = BIT 2. XACT\_FLAGS = fBeginXact 3. 7FRESERVEDBIT 4. ISOLATION\_LEVEL = BYTE 5. XACT\_NAME = B\_VARBYTE 6. BEGIN\_XACT\_NAME = B\_VARBYTE 7. RequestPayload = XACT\_NAME 8. XACT\_FLAGS 9. [ISOLATION\_LEVEL 10. 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\_LEVEL SHOULD NOT be present.  Specifying ISOLATION\_LEVEL allows the isolation level to remain in effect for the session, once the transaction ends.  If fBeginXact is 0, BEGIN\_XACT\_NAME SHOULD NOT be present. If fBeginXact is 1, BEGIN\_XACT\_NAME can be nonempty.  If fBeginXact is 1, a new transaction MUST be started. If BEGIN\_XACT\_NAME is nonempty, the new transaction MUST be given the specified name.  If fBeginXact is 1, and the ROLLBACK only rolled back to a savepoint, the 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:  1. XACT\_SAVEPOINT\_NAME = B\_VARBYTE 2. 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. |

### 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](#Section_7cfc401bf9b6404da44722e21e593742). The corresponding message types that use token-based packet data streams are identified in the table in section [2.2.4](#Section_dc3a08548230482fbbb9d94a3b905a26).

#### ALTMETADATA

**Token Stream Name:**

1. ALTMETADATA

**Token Stream Function:**

Describes the data type, length, and name of column data that result from a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) that generates totals.

**Token Stream Comments:**

The token value is 0x88.

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**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e). ALTMETADATA and the corresponding ALTROW MUST be in the same [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d).

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.

**Note**  ALTMETADATA was deprecated in TDS 7.4.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. Count = USHORT
3. Id = USHORT
4. ByCols = UCHAR
5. Op = BYTE
6. Operand = USHORT
7. UserType = USHORT/ULONG; (changed to ULONG in TDS 7.2)
8. fNullable = BIT
9. fCaseSen = BIT
10. usUpdateable = 2BIT ; 0 = ReadOnly
11. ; 1 = Read/Write
12. ; 2 = Unused
13. fIdentity = BIT
14. fComputed = BIT ; (introduced in TDS 7.2)
15. usReservedODBC = 2BIT
16. fFixedLenCLRType = BIT ; (introduced in TDS 7.2)
17. usReserved = 7BIT
18. Flags = fNullable
19. fCaseSen
20. usUpdateable
21. fIdentity
22. (FRESERVEDBIT / fComputed)
23. usReservedODBC
24. (FRESERVEDBIT / fFixedLenCLRType)
25. usReserved
26. NumParts = BYTE ; (introduced in TDS 7.2)
27. PartName = US\_VARCHAR ; (introduced in TDS 7.2)
28. TableName = US\_VARCHAR ; (removed in TDS 7.2)
29. /
30. (NumParts
31. 1\*PartName) ; (introduced in TDS 7.2)
32. ColName = B\_VARCHAR
33. ColNum = USHORT
34. ComputeData = Op
35. Operand
36. UserType
37. Flags
38. TYPE\_INFO
39. [TableName]
40. ColName

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

**Token Stream Definition:**

1. ALTMETADATA = TokenType
2. Count
3. Id
4. ByCols
5. \*(<ByCols> ColNum)
6. 1\*ComputeData

**Token Stream Parameter Details:**

| Parameter | Description |
| --- | --- |
| TokenType | ALTMETADATA\_TOKEN[<33>](#Appendix_A_33" \o "Product behavior note 33) |
| 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.   1. AOPSTDEV = %x30 ; Standard deviation (STDEV) 2. AOPSTDEVP = %x31 ; Standard deviation of the population (STDEVP) 3. AOPVAR = %x32 ; Variance (VAR) 4. AOPVARP = %x33 ; Variance of population (VARP) 5. AOPCNT = %x4B ; Count of rows (COUNT) 6. AOPSUM = %x4D ; Sum of the values in the rows (SUM) 7. AOPAVG = %x4F ; Average of the values in the rows (AVG) 8. AOPMIN = %x51 ; Minimum value of the rows (MIN) 9. 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](#Section_bbc22f15e1a04338a169c79819d39b1c). 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](#Section_58880b9f381c43b2bf8b0727a98c4f4c):   * fNullable is a bit flag, 1 if the column is nullable. * fCaseSen * 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. |

#### ALTROW

**Token Stream Name:**

1. 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**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d)). This provides the mechanism for matching row data with correct SQL statements. ALTROW and the corresponding ALTMETADATA MUST be in the same [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d).
* **Note**  ALTROW was deprecated in TDS 7.4.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. Id = USHORT
3. Data = TYPE\_VARBYTE
4. ComputeData = Data

**Token Stream Definition:**

1. ALTMETADATA = TokenType
2. Id
3. 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[<34>](#Appendix_A_34" \o "Product behavior note 34) |
| 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**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e). |
| 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. |

#### COLINFO

**Token Stream Name:**

1. COLINFO

**Token Stream Function:**

Describes the column information in browse mode [[MSDN-BROWSE]](https://go.microsoft.com/fwlink/?LinkId=140931), 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:**

1. TokenType = BYTE
2. Length = USHORT
3. ColNum = BYTE
4. TableNum = BYTE
5. Status = BYTE
6. ColName = B\_VARCHAR
7. ColProperty = ColNum
8. TableNum
9. Status
10. [ColName]

The ColInfo element is repeated for each column in the [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d).

**Token Stream Definition:**

1. COLINFO = TokenType
2. Length
3. 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. |

#### COLMETADATA

**Token Stream Name:**

1. COLMETADATA

**Token Stream Function:**

Describes the [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d) for interpretation of following ROW [**data streams**](#gt_151643ce-fb5e-460e-8bdf-dc10bbd1950e).

**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:**

1. TokenType = BYTE
2. Count = USHORT
3. UserType = USHORT/ULONG; (Changed to ULONG in TDS 7.2)
4. fNullable = BIT
5. fCaseSen = BIT
6. usUpdateable = 2BIT ; 0 = ReadOnly
7. ; 1 = Read/Write
8. ; 2 = Unused
9. fIdentity = BIT
10. fComputed = BIT ; (introduced in TDS 7.2)
11. usReservedODBC = 2BIT ; (only exists in TDS 7.3.A and below)
12. fSparseColumnSet = BIT ; (introduced in TDS 7.3.B)
13. fEncrypted = BIT ; (introduced in TDS 7.4)
14. usReserved3 = BIT ; (introduced in TDS 7.4)
15. fFixedLenCLRType = BIT ; (introduced in TDS 7.2)
16. usReserved = 4BIT
17. fHidden = BIT ; (introduced in TDS 7.2)
18. fKey = BIT ; (introduced in TDS 7.2)
19. fNullableUnknown = BIT ; (introduced in TDS 7.2)
20. Flags = fNullable
21. fCaseSen
22. usUpdateable
23. fIdentity
24. (FRESERVEDBIT / fComputed)
25. usReservedODBC
26. (FRESERVEDBIT / fFixedLenCLRType)
27. (usReserved / (FRESERVEDBIT fSparseColumnSet fEncrypted usReserved3))
28. ; (introduced in TDS 7.4)
29. (FRESERVEDBIT / fHidden)
30. (FRESERVEDBIT / fKey)
31. (FRESERVEDBIT / fNullableUnknown)
32. NumParts = BYTE ; (introduced in TDS 7.2)
33. PartName = US\_VARCHAR ; (introduced in TDS 7.2)
34. TableName = NumParts
35. 1\*PartName
36. ColName = B\_VARCHAR
37. BaseTypeInfo = TYPE\_INFO ; (BaseTypeInfo introduced in TDS 7.4)
38. EncryptionAlgo = BYTE ; (EncryptionAlgo introduced in TDS 7.4)
39. AlgoName = B\_VARCHAR ; (introduced in TDS 7.4)
40. EncryptionAlgoType = BYTE ; (introduced in TDS 7.4)
41. NormVersion = BYTE ; (introduced in TDS 7.4)
42. Ordinal = USHORT ; (introduced in TDS 7.4)
43. CryptoMetaData = Ordinal ; (CryptoMetaData introduced in TDS 7.4)
44. UserType
45. BaseTypeInfo
46. EncryptionAlgo
47. [AlgoName]
48. EncryptionAlgoType
49. NormVersion
50. EkValueCount = USHORT ; (introduced in TDS 7.4)
51. CekTable = EkValueCount ; (introduced in TDS 7.4)
52. \*EK\_INFO ; (introduced in TDS 7.4)
53. ColumnData = UserType
54. Flags
55. TYPE\_INFO
56. [TableName]
57. [CryptoMetaData]
58. ColName
59. NoMetaData = %xFF %xFF

The **TableName** element is specified only if a text, ntext, or image column is included in the result set.

**Token Stream Definition:**

1. COLMETADATA = TokenType
2. Count
3. CekTable
4. NoMetaData / (1\*ColumnData)

**Token Stream Parameter Details:**

| 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 [2.2.6.6](#Section_619c43b694954a589e49a4950db245b3) 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 | The size of the Flags parameter is always fixed at 16 bits regardless of the TDS version. Each of the 16 bits of the Flags parameter is interpreted based on the TDS version negotiated during login. Bit flags, in [least significant bit order](#Section_bbc22f15e1a04338a169c79819d39b1c):   * fNullable is a bit flag. Its value is 1 if the column is nullable. * 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. * 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 [**common language runtime user-defined type (CLR UDT)**](#gt_5f1d976e-cd4b-4a78-a6a1-d0bdb0aa0360). * fSparseColumnSet, introduced in TDS version 7.3.B, is a bit flag. Its value is 1 if the column is the special XML column for the sparse column set. For information about using column sets, see [[MSDN-ColSets]](https://go.microsoft.com/fwlink/?LinkId=128616). * fEncrypted is a bit flag. Its value is 1 if the column is encrypted transparently and has to be decrypted to view the plaintext value. This flag is valid when the column encryption feature is negotiated between client and server and is turned on. * 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.[<35>](#Appendix_A_35" \o "Product behavior note 35) * 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. It contains the table name length and table name. This exists only for text, ntext, and image columns. It specifies how many parts will be returned and then repeats PartName once for each NumParts. |
| ColName | The column name. It contains the column name length and column name. |
| BaseTypeInfo | The TYPEINFO for the plaintext data. |
| EkValueCount | The size of CekTable. It represents the number of entries in CekTable. |
| CekTable | A table of various encryption keys that are used to secure the plaintext data. It contains one row for each encryption key. Each row can have multiple encryption key values, and each value represents the cipher text of the same encryption key that is secured by using a different master key. The size of this table is determined by EkValueCount. This table MUST be sent when COLUMNENCRYPTION is negotiated by client and server and is turned on. |
| EncryptionAlgo | A byte that describes the encryption algorithm that is used.  If EncryptionAlgo is set to 1, the algorithm that is used is AEAD\_AES\_256\_CBC\_HMAC\_SHA512, as described in [[IETF-AuthEncr]](https://go.microsoft.com/fwlink/?LinkId=524322) section 5.4. Other values are reserved for future use. |
| AlgoName | Reserved for future use. |
| EncryptionAlgoType | A field that describes the encryption algorithm type. Available values are defined as follows:  1 = Deterministic encryption.  2 = Randomized encryption. |
| NormVersion | The normalization version to which plaintext data MUST be normalized. Version numbering starts at 0x01. |
| Ordinal | Where the encryption key information is located in CekTable. Ordinal starts at 0. |
| CryptoMetaData | This describes the encryption metadata for a column. It contains the ordinal, the UserType, the TYPE\_INFO (BaseTypeInfo) for the plaintext value, the encryption algorithm that is used, the algorithm name literal, the encryption algorithm type, and the normalization version. |
| NoMetaData | This notifies client that no metadata will follow the COLMETADATA token. When fNoMetadata is set to 1, client notifies server that it has already cached the metadata from a previous RPC Request (section 2.2.6.6), and server sends no metadata.[<36>](#Appendix_A_36" \o "Product behavior note 36) |

#### DONE

**Token Stream Name:**

1. DONE

**Token Stream Function:**

Indicates the completion status of a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d).

**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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7), DONEPROC and DONEINPROC tokens are used in place of DONE tokens.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. Status = USHORT
3. CurCmd = USHORT
4. 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:**

1. DONE = TokenType
2. Status
3. CurCmd
4. 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.[<37>](#Appendix_A_37" \o "Product behavior note 37) * 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. * 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d), 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.[<38>](#Appendix_A_38" \o "Product behavior note 38) |

#### DONEINPROC

**Token Stream Name:**

1. DONEINPROC

**Token Stream Function:**

Indicates the completion status of a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) within a [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7).

**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:**

1. TokenType = BYTE
2. Status = USHORT
3. CurCmd = USHORT
4. 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:**

1. DONEINPROC = TokenType
2. Status
3. CurCmd
4. DoneRowCount

**Token Stream Parameter Details:**

| 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.[<39>](#Appendix_A_39" \o "Product behavior note 39) * 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d), 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. |

#### DONEPROC

**Token Stream Name:**

1. DONEPROC

**Token Stream Function:**

Indicates the completion status of a [**stored procedure**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7). This is also generated for stored procedures executed through [**SQL statements**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d).

**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:**

1. TokenType = BYTE
2. Status = USHORT
3. CurCmd = USHORT
4. 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:**

1. DONEPROC = TokenType
2. Status
3. CurCmd
4. 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.[<40>](#Appendix_A_40" \o "Product behavior note 40) * 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d), 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. |

#### ENVCHANGE

**Token Stream Name:**

1. ENVCHANGE

**Token Stream Function:**

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

**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**](#gt_4553803e-9d8d-407c-ad7d-9e65e01d6eb3) 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:**

1. TokenType = BYTE
2. Length = USHORT
3. Type = BYTE
4. EnvValueData = Type
5. NewValue
6. [OldValue]

**Token Stream Definition:**

1. ENVCHANGE = TokenType
2. Length
3. 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.   * 1: Database * 2: Language * 3: Character set * 4: Packet size * 5: [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) data sorting local id * 6: Unicode data sorting comparison flags * 7: SQL Collation * 8: Begin Transaction (described in [[MSDN-BEGIN]](https://go.microsoft.com/fwlink/?LinkId=144544)) * 9: Commit Transaction (described in [[MSDN-COMMIT]](https://go.microsoft.com/fwlink/?LinkId=144542)) * 10: Rollback Transaction * 11: Enlist DTC Transaction * 12: Defect Transaction * 13: Real Time Log Shipping * 15: Promote Transaction * 16: Transaction Manager Address[<41>](#Appendix_A_41" \o "Product behavior note 41) * 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 |
| 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.  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 always reflects 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, and 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]](https://go.microsoft.com/fwlink/?LinkId=144543).
* 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 can be sent back to a client running TDS 7.4 or later regardless of whether the fReadOnlyIntent bit is set in the preceding LOGIN7 record. If a client is running TDS 7.1 to 7.3, type 20 can be sent only if the fReadOnlyIntent bit is set in the preceding LOGIN7 record.

#### ERROR

**Token Stream Name:**

1. ERROR

**Token Stream Function:**

Used to send an error message to the client.

**Token Stream Comments:**

* The token value is 0xAA.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. Length = USHORT
3. Number = LONG
4. State = BYTE
5. Class = BYTE
6. MsgText = US\_VARCHAR
7. ServerName = B\_VARCHAR
8. ProcName = B\_VARCHAR
9. 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:**

1. ERROR = TokenType
2. Length
3. Number
4. State
5. Class
6. MsgText
7. ServerName
8. ProcName
9. 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.[<42>](#Appendix_A_42" \o "Product behavior note 42) |
| 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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) 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 begin at 1. If the line number is not applicable to the message, the value of LineNumber is 0. |

| Class level | Description |
| --- | --- |
| 0-9 | Informational messages that return status information or report errors that are not severe.[<43>](#Appendix_A_43" \o "Product behavior note 43) |
| 10 | Informational messages that return status information or report errors that are not severe.[<44>](#Appendix_A_44" \o "Product behavior note 44) |
| 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**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) 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 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d), the ERROR token is sent before the DONE token for the SQL statement, and such DONE token is sent with the error bit set.

#### FEATUREEXTACK

**Token Stream Name:**

1. FEATUREEXTACK

**Token Stream Function:**

Introduced in TDS 7.4, FEATUREEXTACK is used to send an optional acknowledge message to the client for features that are 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:**

1. TokenType = BYTE
2. FeatureId = BYTE
3. FeatureAckDataLen = DWORD
4. FeatureAckData = \*BYTE
5. TERMINATOR = %xFF ; signal of end of feature ack data
6. FeatureAckOpt = (FeatureId
7. FeatureAckDataLen
8. FeatureAckData)
9. /
10. TERMINATOR

**Token Stream Definition:**

1. FEATUREEXTACK = TokenType
2. 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. The features available to use by a FeatureId are defined in the following table. |
| FeatureAckDataLen | The length of FeatureAckData, in bytes. |
| FeatureAckData | The acknowledge data of a specific feature. Each feature SHOULD define its own data format in the FEATUREEXTACK token if it is selected to acknowledge the feature. |

The following table describes the FeatureExtAck feature option and description.

| FeatureId | FeatureExtData Description |
| --- | --- |
| %0x00 | Reserved. |
| %0x01  (SESSIONRECOVERY)  (introduced in TDS 7.4) | Session Recovery feature. Content is defined as follows:   1. InitSessionStateData = SessionStateDataSet 2. FeatureAckData = InitSessionStateData   SessionStateDataSet is described in section [2.2.7.20](#Section_626fbe19f3564599ba17c70f44005106). 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)[<45>](#Appendix_A_45" \o "Product behavior note 45) | Whenever a login response stream is sent for a TDS connection whose login request includes 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:   1. Nonce = 32BYTE 2. Signature = 32BYTE 3. FeatureAckData = Nonce 4. Signature   Nonce: The client-specified nonce in PRELOGIN.  Signature: The HMAC-SHA-256 [[RFC6234]](https://go.microsoft.com/fwlink/?LinkId=328921) of the client-specified nonce, using the session key retrieved from the [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) context as the shared secret.  When the bFedAuthLibrary is Security Token, the format is as follows:   1. Nonce = 32BYTE 2. 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. |
| %0x04  (COLUMNENCRYPTION)  (introduced in TDS 7.4) | The presence of the COLUMNENCRYPTION FeatureExt SHOULD[<46>](#Appendix_A_46" \o "Product behavior note 46) indicate that the client is capable of performing cryptographic operations on data. The feature data is described as follows:   1. Length = DWORD 2. COLUMNENCRYPTION\_VERSION = BYTE 3. FeatureData = Length 4. COLUMNENCRYPTION\_VERSION 5. \*EnclaveType   COLUMNENCRYPTION\_VERSION: This field defines the cryptographic protocol version that the client understands. The values of this field are as follows:   * 1 = The client supports column encryption without [**enclave computations**](#gt_6fe5534f-5cd8-4ab6-aba4-637a4344eda0). * 2 = The client MAY[<47>](#Appendix_A_47" \o "Product behavior note 47) support column encryption when encrypted data require enclave computations.   EnclaveType: This field is a string that MAY[<48>](#Appendix_A_48" \o "Product behavior note 48) be populated by the server and used by the client to identify the type of [**enclave**](#gt_ef41e9e0-3e5d-432f-96d6-39515bdc5340) that the server is configured to use. During login for the initial connection, the client can request COLUMNENCRYPTION with the Length as 1, and either 1 or 2 as the value of COLUMNENCRYPTION\_VERSION, and no **EnclaveType**. When the client sends COLUMNENCRYPTION\_VERSION as 2 and no **EnclaveType**, the server MUST return COLUMNENCRYPTION\_VERSION as 2 together with the value of **EnclaveType**, if the server supports this column encryption version. If **EnclaveType** is not returned and the column encryption version is returned as 2, the client driver MUST raise an error. |
| %0x05  (GLOBALTRANSACTIONS)[<49>](#Appendix_A_49" \o "Product behavior note 49) | Whenever a login response stream is sent for a TDS connection whose login request includes a GLOBALTRANSACTIONS FeatureExt token, the server login response message stream can optionally include a FEATUREEXTACK token by including the GLOBALTRANSACTIONS FeatureId in the FEATUREEXTACK token stream. The corresponding FeatureAckData MUST then include a flag that indicates whether the server supports [**Global Transactions**](#gt_57552b13-b14e-4601-9621-500ce3297d15). The FeatureAckData format is as follows:   1. IsEnabled = BYTE 2. FeatureAckData = IsEnabled   IsEnabled: Specifies whether the server supports Global Transactions. The values of this field are as follows:   * 0 = The server does not support Global Transactions. * 1 = The server supports Global Transactions. |
| %0x08  (AZURESQLSUPPORT)  (introduced in TDS 7.4) | The presence of the AZURESQLSUPPORT FeatureExt indicates whether failover partner login with read-only intent to Azure SQL Database MAY[<50>](#Appendix_A_50" \o "Product behavior note 50) be supported. For information about failover partner, see [[MSDOCS-DBMirror]](https://go.microsoft.com/fwlink/?linkid=874052).  Whenever a login response stream is sent for a TDS connection whose login request includes an AZURESQLSUPPORT FeatureExt token, the server login response message stream can optionally include a FEATUREEXTACK token by setting the corresponding feature switch in Azure SQL Database. If it is included, the FEATUREEXTACK token stream MUST include the AZURESQLSUPPORT FeatureId.   1. FeatureAckData = BYTE   BYTE: The Bit 0 flag specifies whether failover partner login with read-only intent is supported. The values of this BYTE are as follows:   * 0 = The server does not support the AZURESQLSUPPORT feature extension. * 1 = The server supports the AZURESQLSUPPORT feature extension. |
| %xFF  (TERMINATOR) | This option signals the end of the FeatureExtAck feature and MUST be the feature's last option. |

#### FEDAUTHINFO

**Token Stream Name:**

1. FEDAUTHINFO

**Token Stream Function:**

Introduced in TDS 7.4, [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) information is returned to the client to be used for generating a Federated Authentication Token during the login process. This token MUST be the only token in a Federated Authentication Information message and MUST NOT be included in any other message type.[<51>](#Appendix_A_51" \o "Product behavior note 51)

**Token Stream Comments:**

* The token value is 0xEE.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. TokenLength = DWORD ; (introduced in TDS 7.4)
3. CountOfInfoIDs = DWORD ; (introduced in TDS 7.4)
4. FedAuthInfoID = BYTE ; (introduced in TDS 7.4)
5. FedAuthInfoDataLen = DWORD ; (introduced in TDS 7.4)
6. FedAuthInfoDataOffset = DWORD ; (introduced in TDS 7.4)
7. FedAuthInfoData = VARBYTES ; (introduced in TDS 7.4)
8. FedAuthInfoOpt = (FedAuthInfoID ; (introduced in TDS 7.4)
9. FedAuthInfoDataLen
10. FedAuthInfoDataOffset)

**Token Stream Definition:**

1. FEDAUTHINFO = TokenType ; (introduced in TDS 7.4)
2. TokenLength
3. CountOfInfoIDs
4. 1\*FedAuthInfoOpt
5. FedAuthInfoData

**Token Stream Parameter Details**

| **Parameter** | **Description** |
| --- | --- |
| TokenType | FEDAUTHINFO\_TOKEN |
| TokenLength | The length of the whole Federated Authentication Information token, not including the size occupied by TokenLength itself. The minimum value for this field is sizeof(DWORD) because the field CountOfInfoIDs MUST be present even if no federated authentication information is sent as part of the token. |
| CountOfInfoIDs | The number of federated authentication information options that are sent in the token. If no FedAuthInfoOpt is sent in the token, this field MUST be present and set to 0. |
| FedAuthInfoID | The unique identifier number for the type of information. |
| FedAuthInfoDataLen | The length of FedAuthInfoData, in bytes. |
| FedAuthInfoDataOffset | The offset at which the federated authentication information data for FedAuthInfoID is present, measured from the address of CountOfInfoIDs. |
| FedAuthInfoData | The actual information data as binary, with the length in bytes equal to FedAuthInfoDataLen. |

The following table describes the FedAuthInfo feature option and description.

| **FedAuthInfoID** | **FedAuthInfoData Description** |
| --- | --- |
| %0x00 | Reserved. |
| %0x01  (STSURL) | A [**Unicode**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) string that represents the token endpoint URL from which to acquire a Federated Authentication Token. |
| %0x02  (SPN) | A Unicode string that represents the Service Principal Name (SPN) to use for acquiring a Federated Authentication Token. SPN is a string that represents the resource in a directory. |

#### INFO

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. Length = USHORT
3. Number = LONG
4. State = BYTE
5. Class = BYTE
6. MsgText = US\_VARCHAR
7. ServerName = B\_VARCHAR
8. ProcName = B\_VARCHAR
9. 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:**

1. INFO = TokenType
2. Length
3. Number
4. State
5. Class
6. MsgText
7. ServerName
8. ProcName
9. 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.[<52>](#Appendix_A_52" \o "Product behavior note 52) |
| 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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) 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. |

#### LOGINACK

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. Length = USHORT
3. Interface = BYTE
4. TDSVersion = DWORD
5. ProgName = B\_VARCHAR
6. MajorVer = BYTE
7. MinorVer = BYTE
8. BuildNumHi = BYTE
9. BuildNumLow = BYTE
10. ProgVersion = MajorVer
11. MinorVer
12. BuildNumHi
13. BuildNumLow

**Token Stream Definition:**

1. LOGINACK = TokenType
2. Length
3. Interface
4. TDSVersion
5. ProgName
6. 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**](#gt_95913fbd-3262-47ae-b5eb-18e6806824b9) 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.[<53>](#Appendix_A_53" \o "Product behavior note 53) |
| 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). |

#### NBCROW

**Token Stream Name:**

1. 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**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d) 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:**

1. TokenType = BYTE
2. TextPointer = B\_VARBYTE
3. Timestamp = 8BYTE
4. Data = TYPE\_VARBYTE
5. NullBitmap = <NullBitmapByteCount> BYTE ; see note on NullBitmapByteCount
6. ColumnData = [TextPointer Timestamp] Data
7. 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 might choose to send a ROW token if there is no byte savings if the result set has no [**nullable columns**](#gt_16dd540d-3913-48e5-9a93-a769e85570d0), 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:

1. Byte 1 Byte 2 Byte 3
2. ----------------------- ----------------------- -----------------------
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

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:**

1. NBCROW = TokenType
2. NullBitmap
3. 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. |

#### OFFSET

**Token Stream Name:**

1. OFFSET

**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:**

1. TokenType = BYTE
2. Identifier = USHORT
3. OffSetLen = USHORT

**Token Stream Definition:**

1. OFFSET = TokenType ; (removed in TDS 7.2)
2. Identifier
3. 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). |

#### ORDER

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. Length = USHORT
3. ColNum = \*USHORT

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

**Token Stream Definition:**

1. ORDER = TokenType
2. Length
3. ColNum

**Token Stream Parameter Details**

| Parameter | Description |
| --- | --- |
| TokenType | ORDER\_TOKEN |
| Length | The total length of the ORDER data stream. |
| ColNum | The column number in the [**result set**](#gt_c8a27238-8ccc-442b-9604-75f74d3e6b3d). |

#### RETURNSTATUS

**Token Stream Name:**

1. RETURNSTATUS

**Token Stream Function:**

Used to send the status value of an [RPC](#Section_26327437aa3c4e969bba73a6e862ba21) 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:**

1. TokenType = BYTE
2. Value = LONG

**Token Stream Definition:**

1. RETURNSTATUS = TokenType
2. 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. |

#### RETURNVALUE

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. ParamName = B\_VARCHAR
3. ParamOrdinal = USHORT
4. Status = BYTE
5. UserType = USHORT/ULONG; (Changed to ULONG in TDS 7.2)
6. fNullable = BIT
7. fCaseSen = BIT
8. usUpdateable = 2BIT ; 0 = ReadOnly
9. ; 1 = Read/Write
10. ; 2 = Unused
11. fIdentity = BIT
12. fComputed = BIT ; (introduced in TDS 7.2)
13. usReservedODBC = 2BIT
14. fFixedLenCLRType = BIT ; (introduced in TDS 7.2)
15. usReserved = 7BIT
16. usReserved2 = 2BIT
17. fEncrypted = BIT ; (introduced in TDS 7.4)
18. usReserved3 = 4BIT
19. Flags = fNullable
20. fCaseSen
21. usUpdateable
22. fIdentity
23. (FRESERVEDBIT / fComputed)
24. usReservedODBC
25. (FRESERVEDBIT / fFixedLenCLRType)
26. (usReserved / (usReserved2 fEncrypted usReserved3))
27. ; (introduced in TDS 7.4)
28. TypeInfo = TYPE\_INFO
29. Value = TYPE\_VARBYTE
30. BaseTypeInfo = TYPE\_INFO ; (BaseTypeInfo introduced in TDS 7.4)
31. EncryptionAlgo = BYTE ; (EncryptionAlgo introduced in TDS 7.4)
32. AlgoName = B\_VARCHAR ; (introduced in TDS 7.4)
33. EncryptionAlgoType = BYTE ; (introduced in TDS 7.4)
34. NormVersion = BYTE ; (introduced in TDS 7.4)
35. CryptoMetaData = UserType ; (CryptoMetaData introduced in TDS 7.4)
36. BaseTypeInfo
37. EncryptionAlgo
38. [AlgoName]
39. EncryptionAlgoType
40. NormVersion

**Token Stream Definition:**

1. RETURNVALUE = TokenType
2. ParamOrdinal
3. ParamName
4. Status
5. UserType
6. Flags
7. TypeInfo
8. CryptoMetadata
9. 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**](#gt_324d32b3-f4f3-41c9-b695-78c498094fb7) 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 0x0000 with the exceptions of TIMESTAMP (0x0050) and alias types (greater than 0x00FF). |
| Flags | These bit flags are described in [least significant bit order](#Section_bbc22f15e1a04338a169c79819d39b1c). All of these bit flags SHOULD be set to zero. For a description of each bit flag, see section [2.2.7.4](#Section_58880b9f381c43b2bf8b0727a98c4f4c).   * fNullable * fCaseSen * usUpdateable * fIdentity * fComputed * usReservedODBC * fFixedLengthCLRType * fEncrypted |
| TypeInfo | The TYPE\_INFO for the message. |
| BaseTypeInfo | TYPE\_INFO for the unencrypted type. |
| EncryptionAlgo | A byte that describes the encryption algorithm that is used. AlgoName is populated with the name of the custom encryption algorithm. For all EncryptionAlgo values other than 0, AlgoName MUST NOT be sent. If EncryptionAlgo is set to 1, the algorithm that is used is AEAD\_AES\_256\_CBC\_HMAC\_SHA512, as described in [[IETF-AuthEncr]](https://go.microsoft.com/fwlink/?LinkId=524322) section 5.4. |
| AlgoName | Algorithm name literal that is used to encrypt the plaintext value. |
| EncryptionAlgoType | A field that describes the encryption algorithm type. Available values are defined as follows:  1 = Deterministic encryption.  2 = Randomized encryption. |
| NormVersion | The normalization version to which plaintext data MUST be normalized. Version numbering starts at 0x01. |
| CryptoMetaData | This describes the encryption metadata for a column. It contains the UserType, the TYPE\_INFO (BaseTypeInfo) for the plaintext value, the encryption algorithm that is used, the algorithm name literal, the encryption algorithm type, and the normalization version. |
| Value | The type-dependent data for the parameter (within TYPE\_VARBYTE). |

#### ROW

**Token Stream Name:**

1. ROW

**Token Stream Function:**

Used to send a complete row, as defined by the [COLMETADATA](#Section_58880b9f381c43b2bf8b0727a98c4f4c) token, to the client.

**Token Stream Comments:**

* The token value is 0xD1.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. TextPointer = B\_VARBYTE
3. Timestamp = 8BYTE
4. Data = TYPE\_VARBYTE
5. ColumnData = [TextPointer Timestamp]
6. Data
7. 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:**

1. ROW = TokenType
2. 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. |

#### SESSIONSTATE

**Token Stream Name:**

1. 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](#Section_3c06f11098bd4d5bb836b1ba66452cb7) or [DONEPROC](#Section_65e24140edea46e5b710209af2016195) 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:**

1. fRecoverable = BIT
3. TokenType = BYTE
4. Length = DWORD
5. SeqNo = DWORD
6. Status = fRecoverable 7FRESERVEDBIT
8. StateId = BYTE
9. StateLen = BYTE ; 0-%xFE
10. /
11. (%xFF DWORD) ; %xFF - %xFFFF

14. SessionStateData = StateId
15. StateLen
16. StateValue
18. SessionStateDataSet = 1\*SessionStateData

**Token Stream Definition:**

1. SESSIONSTATE = TokenType
2. Length
3. SeqNo
4. Status
5. SessionStateDataSet

**Token Stream Parameter Details**

| 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. |

#### SSPI

**Token Stream Name:**

1. SSPI

**Token Stream Function:**

The SSPI token returned during the login process.

**Token Stream Comments:**

* The token value is 0xED.

**Token Stream-Specific Rules:**

1. TokenType = BYTE
2. SSPIBuffer = US\_VARBYTE

**Token Stream Definition:**

1. SSPI = TokenType
2. SSPIBuffer

**Token Stream Parameter Details:**

| Parameter | Description |
| --- | --- |
| TokenType | SSPI\_TOKEN |
| SSPIBuffer | The length of the SSPIBuffer and the SSPI buffer using B\_VARBYTE format. |

#### TABNAME

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. Length = USHORT
3. NumParts = BYTE ; (introduced in TDS 7.1 Revision 1)
4. PartName = US\_VARCHAR ; (introduced in TDS 7.1 Revision 1)
5. TableName = US\_VARCHAR ; (removed in TDS 7.1 Revision 1)
6. /
7. (NumParts
8. 1\*PartName) ; (introduced in TDS 7.1 Revision 1)
9. AllTableNames = TableName

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

**Token Stream Definition:**

1. TABNAME = TokenType
2. Length
3. 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. |

#### TVP\_ROW

**Token Stream Name:**

1. 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:**

1. TokenType = BYTE
2. TvpColumnData = TYPE\_VARBYTE
3. 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](#Section_0dfc5367a3884c929ba44d28e775acbc)).

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

**Token Stream Definition:**

1. TVP\_ROW = TokenType
2. AllColumnData

**Token Stream Parameter Details:**

| 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. |

# Protocol Details

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

## Common Details

As described in section [1.3](#Section_893fcc7e8a394b3c815a773b7b982c50), 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.

### Abstract Data Model

See sections [3.2.1](#Section_0e7007a13da74323914828d89ec2ac48) and [3.3.1](#Section_1457f2976d7d4b47a4bef35d273b8cae) for the abstract data model of the client and server, respectively.

### Timers

See section [3.2.2](#Section_8313b90eeee347e1844173b3b5df1e39) for a description of the client timer used and section [3.3.2](#Section_25a86907cd7a44a39c923b9a9c0e127f) for a description of the server timer used.

### Initialization

None.

### Higher-Layer Triggered Events

For information about higher-layer triggered events, see section [3.2.4](#Section_e7af6058672941438815f82a475f9ba6) for a TDS client and section [3.3.4](#Section_6af826c5a9574438a7c0b3bbd309c69a) for a TDS server.

### Message Processing Events and Sequencing Rules

The following series of sequence diagrams illustrate the messages that can be exchanged between client and server. See sections [3.2.5](#Section_bd3a16dd75b64546933059c3ef44d50e) and [3.3.5](#Section_8d3c50da023f4bc5959c60f38ed22825) for specific client and server details regarding message processing events and sequencing rules.

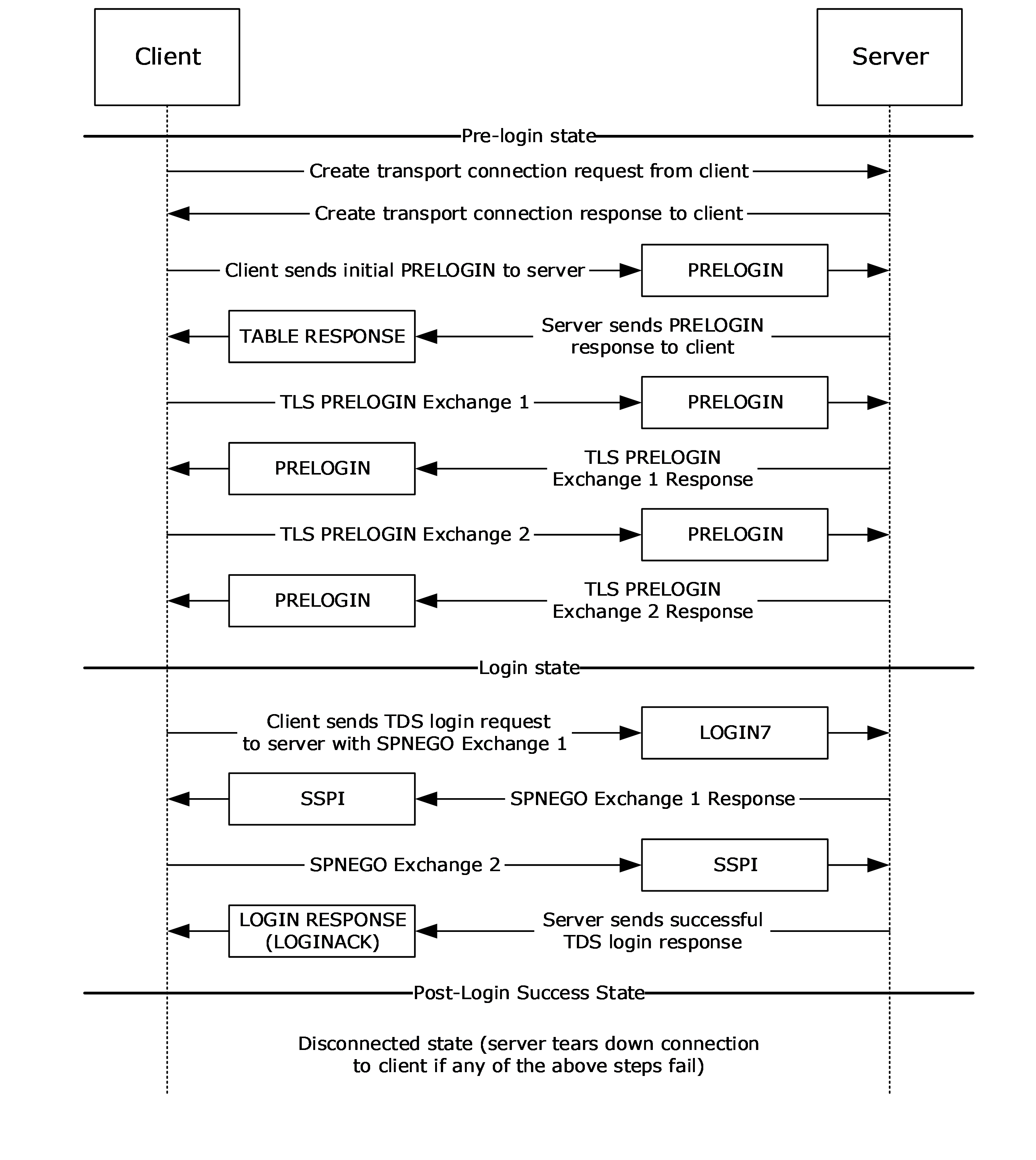


Figure 3: Pre-login to post-login sequence

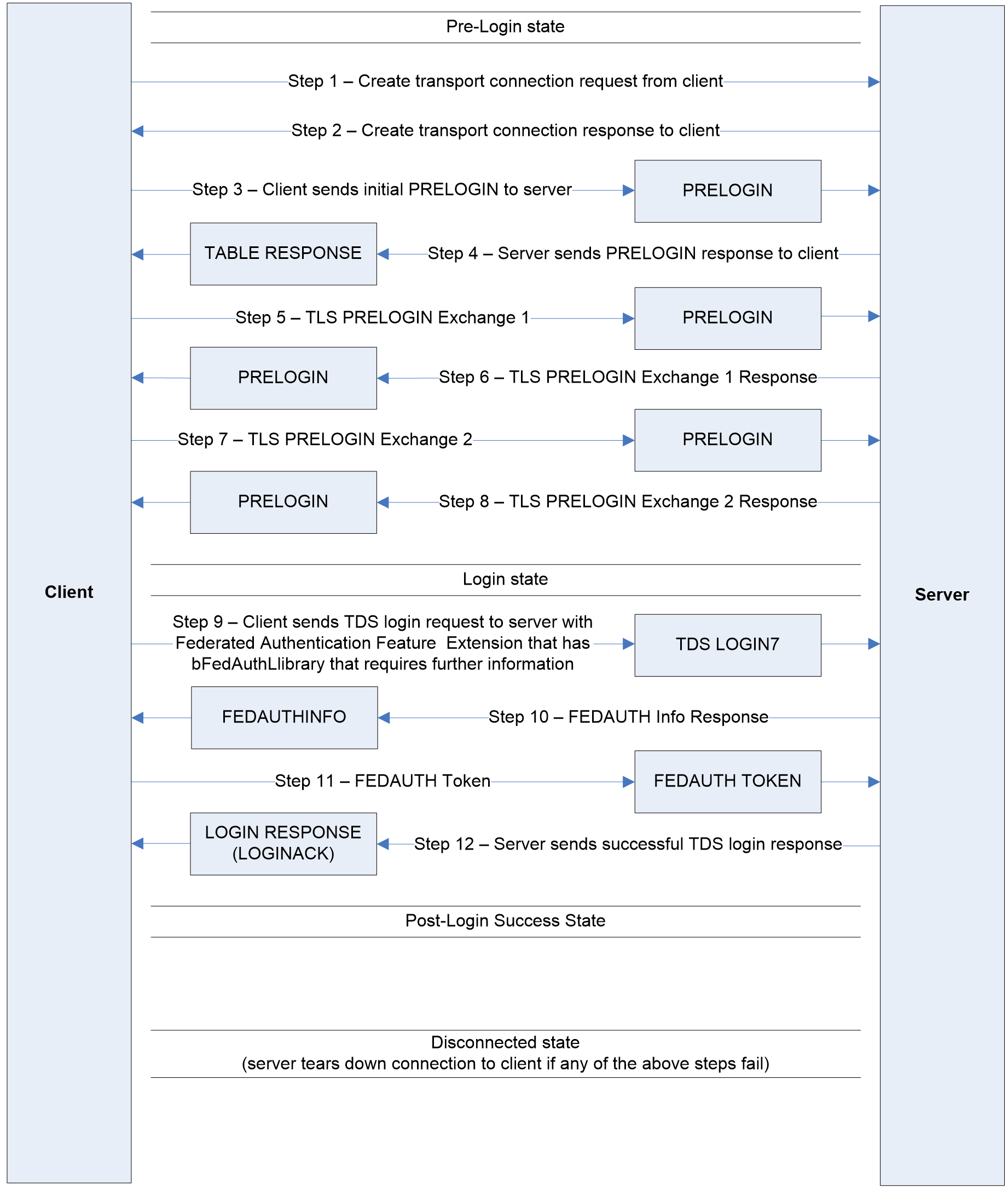


Figure 4: Pre-login to post-login sequence with federated authentication that uses a client library that requires additional information from a server to generate a federated authentication token

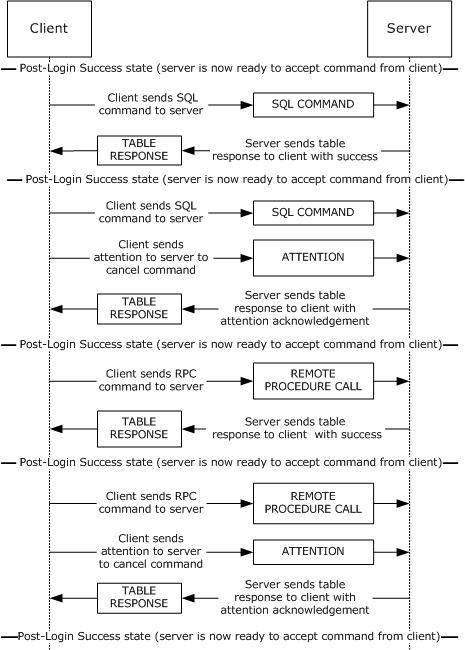


Figure 5: SQL command and RPC sequence

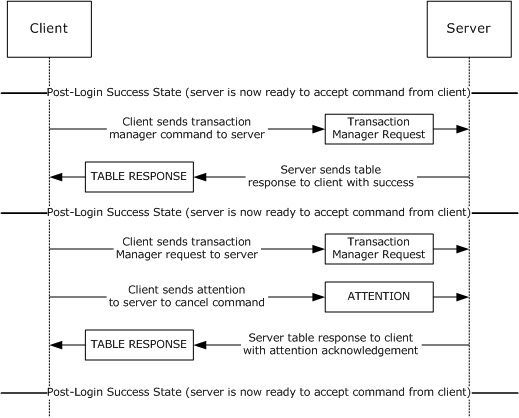


Figure 6: Transaction manager request sequence

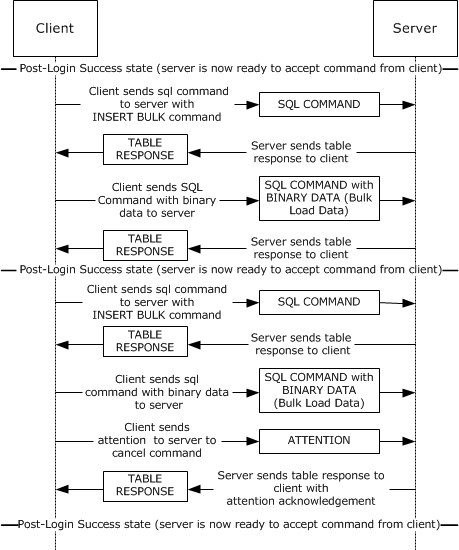


Figure 7: Bulk insert sequence

### Timer Events

See sections [3.2.6](#Section_030af85119fd4a9ca7df05b637f7d888) and [3.3.6](#Section_c0d2d1a29c3d4eb68a460a7dab3b3f56) for the timer events of the client and server, respectively.

### 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](#Section_97d920799ea14a7ba915fa7e8045d88b) and [3.3.7](#Section_388d89cc00a34f57bc22624e251e68f3) for the other local events of the client and server, respectively.

## Client Details

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

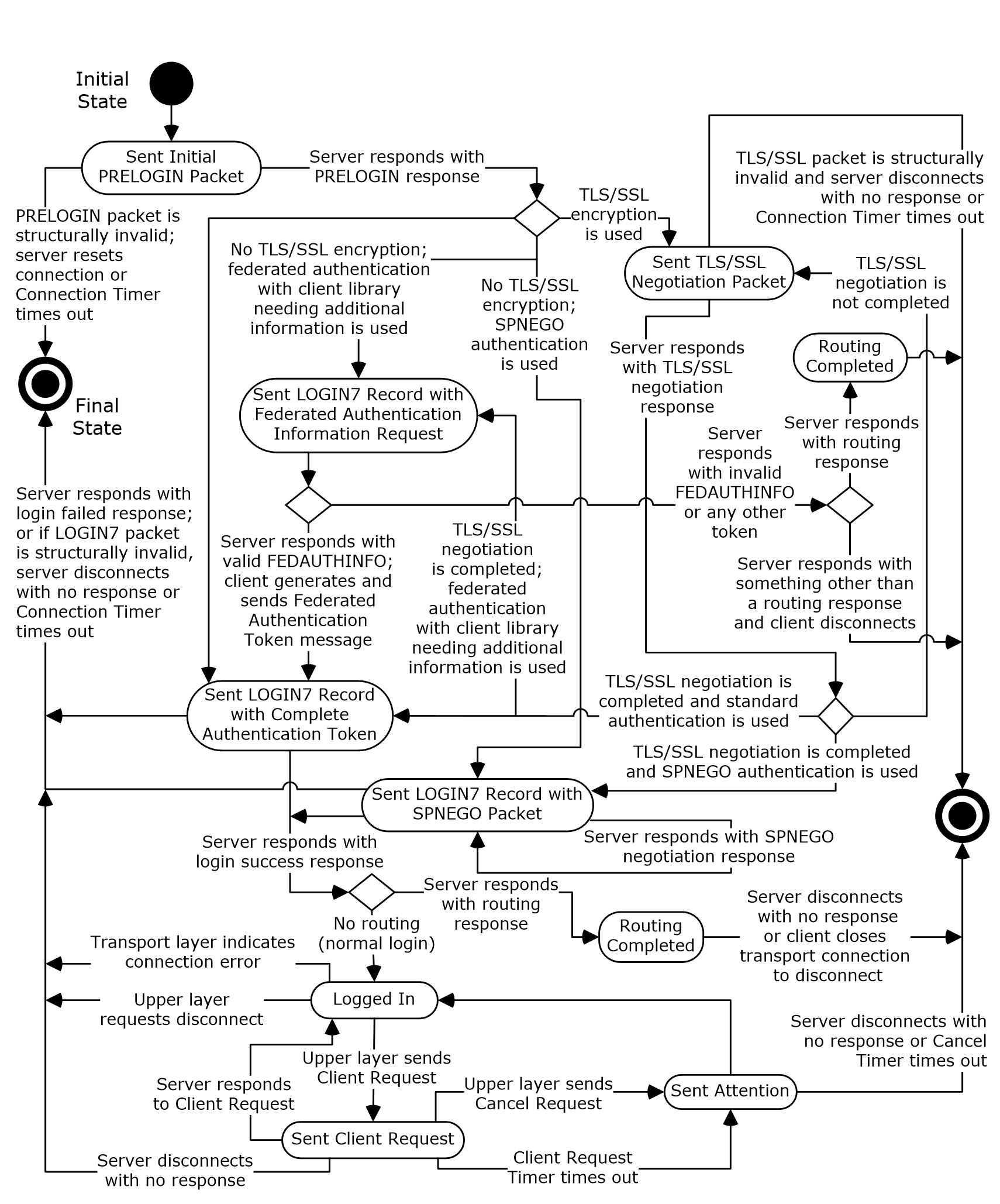


Figure 8: TDS client state machine

### Abstract Data Model

This section describes a conceptual model of 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](#Section_cc823ca848674387819dcc5c19da5732)
* [Sent TLS/SSL Negotiation Packet State](#Section_d62e225bd8654ccc8f73de1ef49e30d4)
* [Sent LOGIN7 Record with Complete Authentication Token State](#Section_ab7a745416a949ef8f208a0de2b31fbb)
* [Sent LOGIN7 Record with SPNEGO Packet State](#Section_c373a90b113d4c5f826ad410365ee883)
* [Sent LOGIN7 Record with Federated Authentication Information Request State](#Section_c272e054279f4b09ac41f9ccf1d1ffde)
* [Logged In State](#Section_e2e7293f55cf43eeb0282d6dea625db6)
* [Sent Client Request State](#Section_c89701263cbc446f9275cc36ba62a2f4)
* [Sent Attention State](#Section_61fb4f9d95ec41718d4bdac4eb16149f)
* [Routing Completed State](#Section_52851226121146c8ac506b7facd08fd9)
* [Final State](#Section_07a0c71e1a7b44558da47e43a09589bc)

### 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).[<54>](#Appendix_A_54" \o "Product behavior note 54)
* 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).[<55>](#Appendix_A_55" \o "Product behavior note 55)

For all three timers, a client can implement a minimum timeout value that is as short as required. 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]](https://go.microsoft.com/fwlink/?LinkId=112180) 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.

### Initialization

None.

### 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.
* 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, Bulk Load, 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**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) 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**](#gt_5aca1780-e1d4-4310-9b28-e8f1caccde61) 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 SMP session, or else an existing SMP session MAY be used.
* Send either SQL Command, Bulk Load, 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 SMP 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 SMP layer.
* Enter the "Sent Attention" state.

### 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](#Section_e5ea85201ea34a75a2a9c17e63e9ee19).

Whenever the TDS client enters either the "Logged In" state or the "[**Final State**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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**](#gt_71dd1dd2-c167-49a8-a5f5-6b0df5c8b48a) (TDS packet type %x04) from the server, it MUST behave as follows, according to the state of the TDS client.

#### 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 [2.2.6.5](#Section_60f5640801884cd58b9025c6f2423868) 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 that contains the authentication scheme that is specified by the user and MUST enter one of the following three states, depending on the message sent:
  + "Sent LOGIN7 record with Complete Authentication Token" state, if a login message that contains either of the following was sent.
    - Standard authentication.
    - FEDAUTH FeatureExt that indicates a client library that does not need any additional information from the server for authentication.
  + "Sent LOGIN7 record with [**SPNEGO**](#gt_bc2f6b5e-e5c0-408b-8f55-0350c24b9838) packet" state, if a Login message with SPNEGO authentication was sent.
  + "Sent LOGIN7 record with Federated Authentication Information Request" state, if a Login message with FEDAUTH FeatureExt that indicates a client library that needs additional information from the server for authentication was sent.

The TDS specification does not prescribe the authentication protocol if SSPI [[SSPI]](https://go.microsoft.com/fwlink/?LinkId=90536) authentication is used. The current implementation of SSPI supports NTLM [[MSDN-NTLM]](https://go.microsoft.com/fwlink/?LinkId=145227) and Kerberos [[RFC4120]](https://go.microsoft.com/fwlink/?LinkId=90458).

* 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### 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 that contains the authentication scheme that is specified by the user. The TDS client then enters one of the following three states, depending on the message sent:
  + "Sent LOGIN7 record with Complete Authentication Token" state, if a Login message that contains either of the following was sent:
    - Standard authentication.
    - FEDAUTH FeatureId that indicates a client library that does not need any additional information from the server for authentication.
  + The "Sent LOGIN7 record with [**SPNEGO**](#gt_bc2f6b5e-e5c0-408b-8f55-0350c24b9838) packet" state, if a Login message with SPNEGO authentication was sent.
  + "Sent LOGIN7 record with Federated Authentication Information Request" state, if a Login message with FEDAUTH FeatureExt that indicates a client library that needs additional information from server for authentication was sent.

The TDS specification does not prescribe the authentication protocol if SSPI [[SSPI]](https://go.microsoft.com/fwlink/?LinkId=90536) authentication or federated authentication is used. The current implementation of SSPI supports NTLM [[MSDN-NTLM]](https://go.microsoft.com/fwlink/?LinkId=145227) and Kerberos [[RFC4120]](https://go.microsoft.com/fwlink/?LinkId=90458).

* If login-only encryption was negotiated as described in section [2.2](#Section_a734db771cb247e4b61a0c9dbd8f960f) 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### 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**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) 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]](https://go.microsoft.com/fwlink/?LinkId=328921) 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 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 or Active Directory Authentication Library (ADAL) [that is, 0x02] 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state.

#### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state.

#### Sent LOGIN7 Record with Federated Authentication Information Request State

If the response received from the server contains a structurally valid Login Response message that contains a Routing ENVCHANGE token in the response after the LOGINACK token, the TDS client MUST enter the "Routing Completed" state.

If the response received from the server contains a structurally valid Login Response message that contains a FEDAUTHINFO token, the TDS client MUST generate a Federated Authentication message, send that Federated Authentication message to the server, and enter the "Sent LOGIN7 record with Complete Authentication Token" state.

If the response received from the server does not contain a structurally valid Login Response message that contains a routing response or a structurally valid FEDAUTHINFO token, the TDS client MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.

#### 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**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) is enabled, the TDS client MUST send the appropriate request to the SMP layer. If the upper layer requests a termination of the connection, the TDS client MUST disconnect from the server and enter the "[**Final State**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state.

#### Sent Attention State

If the response is structurally valid and it does not acknowledge the Attention as described in section [2.2.1.7](#Section_dc28579f49b14a789c5f63fbda002d2e), 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 2.2.1.7, 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](#Section_e7af6058672941438815f82a475f9ba6) or query timeout as described in section [3.2.2](#Section_8313b90eeee347e1844173b3b5df1e39)), 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state.

#### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state. The original connection cannot be used for any other purpose after the Routing ENVCHANGE token is read and the response is drained.

#### Final State

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

### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

## Server Details

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

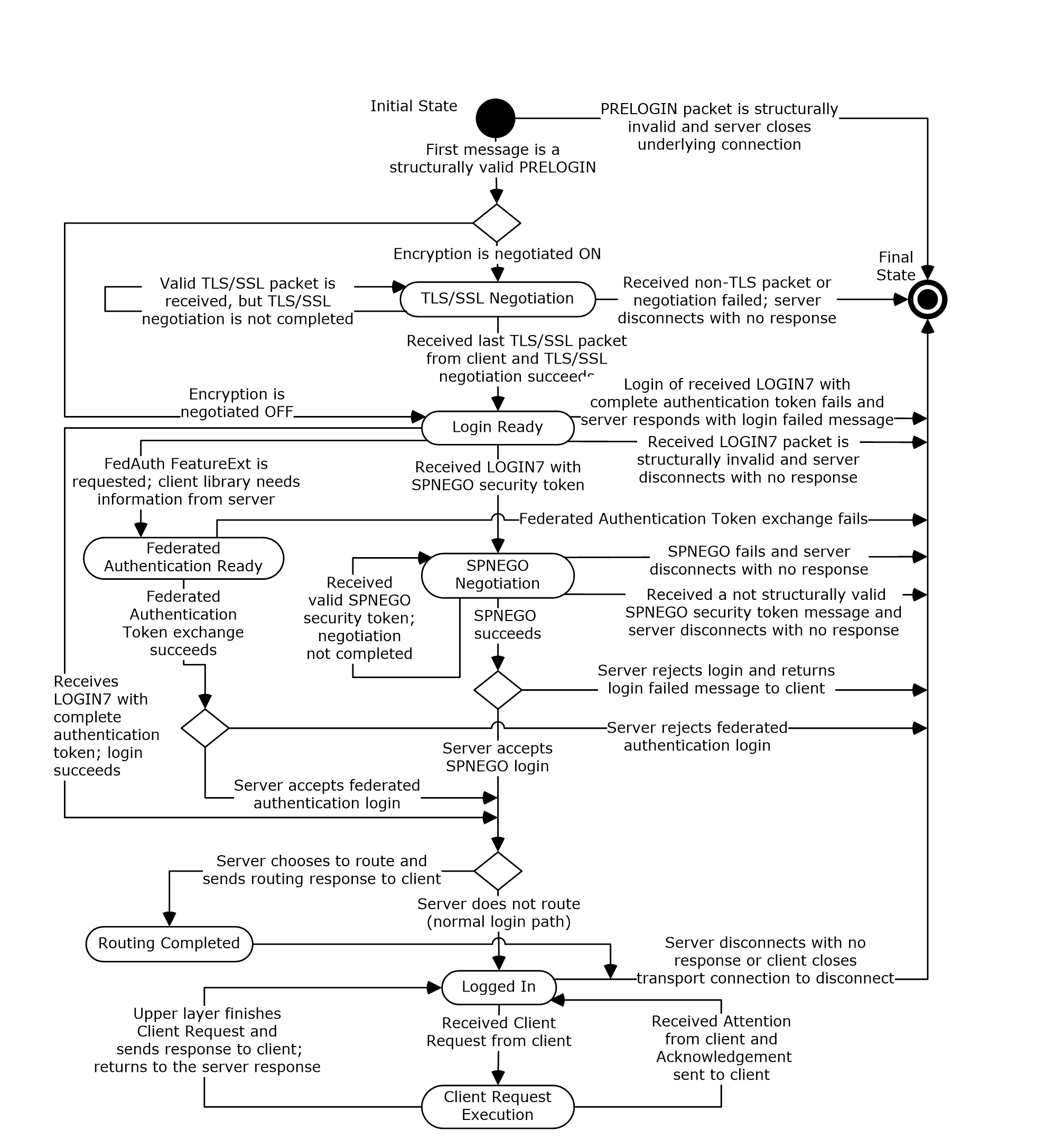


Figure 9: TDS server state machine

### Abstract Data Model

This section describes a conceptual model of 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](#Section_8bfcbcd21baf47928fff86a63a9e90fa)
* [TLS/SSL Negotiation State](#Section_ef1c4791413f4ec7ad475810a514db94)
* [Login Ready State](#Section_1d82a68b57c246e984baed41f7bd0c7c)
* [SPNEGO Negotiation State](#Section_7b617d06e13845f3bcea35619b14ac72)
* [Federated Authentication Ready State](#Section_f6374988cc5446dbadbeb7742b8600e6)
* [Logged In State](#Section_7c37e9c36c6544dfbbd81e8fc11a618d)
* [Client Request Execution State](#Section_21c9c608c6bc456fb7eb9f09a90bd282)
* [Routing Completed State](#Section_5d9e2ee4b6f24333a9b8944109b78c80)
* [Final State](#Section_f9157109df6b4f23ab9135a5ca43260d)

### 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](#Section_8d0d4f5624a145ffb0bacc7d17517e82).

### Initialization

The server MUST establish a listening endpoint based on one of the transport protocols described in section [2.1](#Section_fd30432f71b2488cb30f19737d76d970). 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](#Section_58886b79ec7542f3bd7be32f327366b6) message.

### 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.

### 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](#Section_a734db771cb247e4b61a0c9dbd8f960f).

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

#### 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](#Section_8d0d4f5624a145ffb0bacc7d17517e82). The TDS server SHOULD close the underlying transport connection, indicate an error to the upper layer, and enter the "[**Final State**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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**](#gt_71dd1dd2-c167-49a8-a5f5-6b0df5c8b48a) (0x04) packet and enter "TLS/SSL Negotiation" state if encryption is negotiated.
* Return to the client a PRELOGIN structure wrapped in a table response (0x04) packet and enter unencrypted "Login Ready" state if encryption is not negotiated.

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](#Section_1d82a68b57c246e984baed41f7bd0c7c).

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.

#### 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### 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](#Section_8d0d4f5624a145ffb0bacc7d17517e82), 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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 a 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:
    - The TDS server MUST use the session key from the federated authentication token to compute the HMAC-SHA-256 [[RFC6234]](https://go.microsoft.com/fwlink/?LinkId=328921) 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.

When bFedAuthLibrary is Active Directory Authentication Library (ADAL) [that is, 0x02], the TDS server MUST validate that no other data was sent as part of the feature extension, that is, that FeatureExt is structurally valid for this library type. Then the TDS server MUST send a FEDAUTHINFO token with data for FedAuthInfoIDs of STSURL and SPN and enter the "Federated Authentication Ready" state. This FEDAUTHINFO Token message SHOULD be used by the client to generate a federated authentication token.

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 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.

#### 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**](#gt_bc2f6b5e-e5c0-408b-8f55-0350c24b9838) [[RFC4178]](https://go.microsoft.com/fwlink/?LinkId=90461) 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**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### Federated Authentication Ready State

This state is used to process the [**federated authentication**](#gt_5ae22a0e-5ff4-441b-80d4-224ef4dd4d19) token that is obtained from the client. The TDS server processes the packet that is received according to the following rules:

* If the packet that is received is a structurally valid Federated Authentication Token message, the TDS server MUST delegate processing of the security token embedded in the packet to the federated authentication layer, using the library that is indicated by the state variable that maintains the value of the bFedAuthLibrary field of the login packet’s FEDAUTH FeatureExt. The federated authentication layer responds with one of two results, and the TDS server continues processing according to the response as follows:
  + SUCCESS: The TDS Server MUST send the Federated Authentication Token to the upper layer (typically, an application that provides database management functions) for authorization. If the upper layer approves the token, the TDS server MUST send a LoginACK message that includes a FEATUREEXTACK token that contains 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 token, then a "Login Failed" ERROR token MUST be sent back to the client, the TDS server MUST close the connection, and the TDS server MUST enter the "Final State" state.
  + ERROR: The server MUST close the underlying transport connection, indicate an error to the upper layer, and enter the "Final State" state.
* If the packet that is received is not a structurally valid Federated Authentication Token message, the TDS server SHOULD 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.

#### Logged In State

If a TDS of type 1, 3, 7, or 14 (see section [2.2.3.1.1](#Section_9b4a463c26344a4bac35bebfff2fb0f7)) 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](#Section_21c9c608c6bc456fb7eb9f09a90bd282) state. If any other TDS types arrive, then the server MUST enter the [Final State](#Section_f9157109df6b4f23ab9135a5ca43260d) state. The TDS server MUST continue to listen for messages from the client while awaiting notification of client request completion from the upper layer.

#### 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](#Section_8d0d4f5624a145ffb0bacc7d17517e82) 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**](#gt_762fe1e3-0979-4402-b963-1e9150de133d) is enabled, all TDS server responses to client request messages MUST be passed through to the SMP layer.
* If any other message type arrives, the server MUST close the connection and enter the "[**Final State**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" state.

#### Routing Completed State

The TDS server SHOULD wait for connection closure initiated by the client and enter the "[**Final State**](#gt_c234b203-b3ef-4c4b-be22-058de5ed087e)" 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.

#### Final State

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

### Timer Events

None.

### Other Local Events

When there is a failure in under-layers, the server SHOULD terminate the [**TDS session**](#gt_276cd76b-c0a2-4f7c-8529-ad0d60aa9592) 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.

# 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.

## Pre-Login Request

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

1. 12 01 00 2F 00 00 01 00 00 00 1A 00 06 01 00 20
2. 00 01 02 00 21 00 01 03 00 22 00 04 04 00 26 00
3. 01 FF 09 00 00 00 00 00 01 00 B8 0D 00 00 01
4. <PacketHeader>
5. <Type>
6. <BYTE>12 </BYTE>
7. </Type>
8. <Status>
9. <BYTE>01 </BYTE>
10. </Status>
11. <Length>
12. <BYTE>00 </BYTE>
13. <BYTE>2F </BYTE>
14. </Length>
15. <SPID>
16. <BYTE>00 </BYTE>
17. <BYTE>00 </BYTE>
18. </SPID>
19. <Packet>
20. <BYTE>01 </BYTE>
21. </Packet>
22. <Window>
23. <BYTE>00 </BYTE>
24. </Window>
25. </PacketHeader>
26. <PacketData>
27. <Prelogin>
28. <TokenType>
29. <BYTE>00 </BYTE>
30. </TokenType>
31. <TokenPosition>
32. <USHORT>00 1A</USHORT>
33. </TokenPosition>
34. <TokenLeng>
35. <USHORT>00 06</USHORT>
36. </TokenLeng>
37. <TokenType>
38. <BYTE>01 </BYTE>
39. </TokenType>
40. <TokenPosition>
41. <USHORT>00 20</USHORT>
42. </TokenPosition>
43. <TokenLeng>
44. <USHORT>00 01</USHORT>
45. </TokenLeng>
46. <TokenType>
47. <BYTE>02 </BYTE>
48. </TokenType>
49. <TokenPosition>
50. <USHORT>00 21</USHORT>
51. </TokenPosition>
52. <TokenLeng>
53. <USHORT>00 01</USHORT>
54. </TokenLeng>
55. <TokenType>
56. <BYTE>03 </BYTE>
57. </TokenType>
58. <TokenPosition>
59. <USHORT>00 22</USHORT>
60. </TokenPosition>
61. <TokenLeng>
62. <USHORT>00 04</USHORT>
63. </TokenLeng>
64. <TokenType>
65. <BYTE>04 </BYTE>
66. </TokenType>
67. <TokenPosition>
68. <USHORT>00 26</USHORT>
69. </TokenPosition>
70. <TokenLeng>
71. <USHORT>00 01</USHORT>
72. </TokenLeng>
73. <TokenType>
74. <BYTE>FF </BYTE>
75. </TokenType>
76. <PreloginData>
77. <BYTES>09 00 00 00 00 00 01 00 B8 0D 00 00 01</BYTE>
78. </PreloginData>
79. </Prelogin>
80. </PacketData>

## Login Request

LOGIN7 stream sent from the client to the server:

1. 10 01 00 90 00 00 01 00 88 00 00 00 02 00 09 72
2. 00 10 00 00 00 00 00 07 00 01 00 00 00 00 00 00
3. E0 03 00 00 E0 01 00 00 09 04 00 00 5E 00 08 00
4. 6E 00 02 00 72 00 00 00 72 00 07 00 80 00 00 00
5. 80 00 00 00 80 00 04 00 88 00 00 00 88 00 00 00
6. 00 50 8B E2 B7 8F 88 00 00 00 88 00 00 00 88 00
7. 00 00 00 00 00 00 73 00 6B 00 6F 00 73 00 74 00
8. 6F 00 76 00 31 00 73 00 61 00 4F 00 53 00 51 00
9. 4C 00 2D 00 33 00 32 00 4F 00 44 00 42 00 43 00
10. <PacketHeader>
11. <Type>
12. <BYTE>10 </BYTE>
13. </Type>
14. <Status>
15. <BYTE>01 </BYTE>
16. </Status>
17. <Length>
18. <BYTE>00 </BYTE>
19. <BYTE>90 </BYTE>
20. </Length>
21. <SPID>
22. <BYTE>00 </BYTE>
23. <BYTE>00 </BYTE>
24. </SPID>
25. <Packet>
26. <BYTE>01 </BYTE>
27. </Packet>
28. <Window>
29. <BYTE>00 </BYTE>
30. </Window>
31. </PacketHeader>
32. <PacketData>
33. <Login7>
34. <Length>
35. <DWORD>88 00 00 00 </DWORD>
36. </Length>
37. <TDSVersion>
38. <DWORD>02 00 09 72 </DWORD>
39. </TDSVersion>
40. <PacketSize>
41. <DWORD>00 10 00 00 </DWORD>
42. </PacketSize>
43. <ClientProgVer>
44. <DWORD>00 00 00 07 </DWORD>
45. </ClientProgVer>
46. <ClientPID>
47. <DWORD>00 01 00 00 </DWORD>
48. </ClientPID>
49. <ConnectionID>
50. <DWORD>00 00 00 00 </DWORD>
51. </ConnectionID>
52. <OptionFlags1>
53. <BYTE>E0 </BYTE>
54. </OptionFlags1>
55. <OptionFlags2>
56. <BYTE>03 </BYTE>
57. </OptionFlags2>
58. <TypeFlags>
59. <BYTE>00 </BYTE>
60. </TypeFlags>
61. <OptionFlags3>
62. <BYTE>00 </BYTE>
63. </OptionFlags3>
64. <ClientTimZone>
65. <LONG>E0 01 00 00 </LONG>
66. </ClientTimZone>
67. <ClientLCID>
68. <DWORD>09 04 00 00 </DWORD>
69. </ClientLCID>
70. <OffsetLength>
71. <ibHostName>
72. <USHORT>5E 00 </USHORT>
73. </ibHostName>
74. <cchHostName>
75. <USHORT>08 00 </USHORT>
76. </cchHostName>
77. <ibUserName>
78. <USHORT>6E 00 </USHORT>
79. </ibUserName>
80. <cchUserName>
81. <USHORT>02 00 </USHORT>
82. </cchUserName>
83. <ibPassword>
84. <USHORT>72 00 </USHORT>
85. </ibPassword>
86. <cchPassword>
87. <USHORT>00 00 </USHORT>
88. </cchPassword>
89. <ibAppName>
90. <USHORT>72 00 </USHORT>
91. </ibAppName>
92. <cchAppName>
93. <USHORT>07 00 </USHORT>
94. </cchAppName>
95. <ibServerName>
96. <USHORT>80 00 </USHORT>
97. </ibServerName>
98. <cchServerName>
99. <USHORT>00 00 </USHORT>
100. </cchServerName>
101. <ibUnused>
102. <USHORT>80 00 </USHORT>
103. </ibUnused>
104. <cbUnused>
105. <USHORT>00 00 </USHORT>
106. </cbUnused>
107. <ibCltIntName>
108. <USHORT>80 00 </USHORT>
109. </ibCltIntName>
110. <cchCltIntName>
111. <USHORT>04 00 </USHORT>
112. </cchCltIntName>
113. <ibLanguage>
114. <USHORT>88 00 </USHORT>
115. </ibLanguage>
116. <cchLanguage>
117. <USHORT>00 00 </USHORT>
118. </cchLanguage>
119. <ibDatabase>
120. <USHORT>88 00 </USHORT>
121. </ibDatabase>
122. <cchDatabase>
123. <USHORT>00 00 </USHORT>
124. </cchDatabase>
125. <ClientID>
126. <BYTES>00 50 8B E2 B7 8F </BYTES>
127. </ClientID>
128. <ibSSPI>
129. <USHORT>88 00 </USHORT>
130. </ibSSPI>
131. <cbSSPI>
132. <USHORT>00 00 </USHORT>
133. </cbSSPI>
134. <ibAtchDBFile>
135. <USHORT>88 00 </USHORT>
136. </ibAtchDBFile>
137. <cchAtchDBFile>
138. <USHORT>00 00 </USHORT>
139. </cchAtchDBFile>
140. <ibChangePassword>
141. <USHORT>88 00 </USHORT>
142. </ibChangePassword>
143. <cchChangePassword>
144. <USHORT>00 00 </USHORT>
145. </cchChangePassword>
146. <cbSSPILong>
147. <LONG>00 00 00 00 </LONG>
148. </cbSSPILong>
149. </OffsetLength>
150. <Data>
151. <BYTES>73 00 6B 00 6F 00 73 00 74 00 6F 00 76 00 31 00 73 00 61 00
152. 4F 00 53 00 51 00 4C 00 2D 00 33 00 32 00 4F 00 44 00 42 00 43 00 </BYTES>
153. </Data>
154. </Login7>
155. </PacketData>

## Login Request with Federated Authentication

LOGIN7 stream sent from client to server, including the Feature Extension block that contains the federated authentication feature:

1. 10 01 08 10 00 00 01 00 08 08 00 00 04 00 00 74
2. 00 10 00 00 00 00 00 07 40 37 00 00 00 00 00 00
3. E0 03 00 10 E0 01 00 00 09 04 00 00 5E 00 0F 00
4. 00 00 00 00 00 00 00 00 7C 00 06 00 88 00 18 00
5. B8 00 04 00 BC 00 04 00 C4 00 00 00 C4 00 00 00
6. 00 15 5D 71 E7 42 00 00 00 00 C4 00 00 00 00 00
7. 00 00 00 00 00 00 44 00 41 00 4E 00 42 00 45 00
8. 4E 00 45 00 44 00 33 00 2D 00 58 00 47 00 5A 00
9. 55 00 4F 00 53 00 51 00 4C 00 43 00 4D 00 44 00
10. 63 00 6C 00 6F 00 75 00 64 00 2E 00 64 00 65 00
11. 76 00 2E 00 6D 00 73 00 63 00 64 00 73 00 2E 00
12. 63 00 6F 00 6D 00 2C 00 31 00 34 00 33 00 35 00
13. C4 00 00 00 4F 00 44 00 42 00 43 00 02 3E 07 00
14. 00 01 E2 06 00 00 74 00 3D 00 45 00 77 00 43 00
15. 51 00 41 00 6A 00 4B 00 6A 00 42 00 77 00 41 00
16. 55 00 53 00 30 00 48 00 6F 00 2F 00 30 00 65 00
17. 31 00 52 00 35 00 4D 00 32 00 77 00 37 00 74 00
18. 69 00 44 00 72 00 36 00 70 00 4D 00 63 00 75 00
19. 77 00 33 00 35 00 6B 00 41 00 41 00 61 00 62 00
20. 76 00 45 00 38 00 6F 00 55 00 45 00 71 00 2F 00
21. 50 00 45 00 75 00 4C 00 37 00 32 00 64 00 54 00
22. 74 00 64 00 4D 00 42 00 43 00 7A 00 46 00 37 00
23. 41 00 64 00 4A 00 6D 00 6E 00 70 00 45 00 68 00
24. 34 00 32 00 65 00 41 00 46 00 57 00 37 00 2F 00
25. 38 00 49 00 6A 00 41 00 53 00 6D 00 78 00 79 00
26. 50 00 30 00 30 00 71 00 59 00 64 00 76 00 65 00
27. 31 00 5A 00 6B 00 45 00 2F 00 58 00 6C 00 63 00
28. 4E 00 79 00 69 00 77 00 6F 00 55 00 76 00 54 00
29. 4E 00 4F 00 6B 00 74 00 42 00 45 00 4C 00 71 00
30. 76 00 37 00 58 00 4A 00 54 00 6F 00 47 00 71 00
31. 56 00 32 00 6F 00 6B 00 51 00 6E 00 2F 00 65 00
32. 63 00 50 00 41 00 78 00 32 00 71 00 6A 00 55 00
33. 57 00 74 00 6B 00 54 00 59 00 56 00 66 00 62 00
34. 72 00 51 00 58 00 78 00 76 00 4E 00 58 00 66 00
35. 69 00 72 00 77 00 47 00 57 00 63 00 43 00 45 00
36. 4B 00 79 00 46 00 64 00 76 00 35 00 62 00 78 00
37. 55 00 75 00 68 00 69 00 49 00 62 00 39 00 42 00
38. 55 00 56 00 54 00 56 00 4B 00 6A 00 57 00 51 00
39. 34 00 78 00 4E 00 46 00 4E 00 6B 00 43 00 33 00
40. 73 00 58 00 71 00 6F 00 46 00 52 00 31 00 49 00
41. 46 00 64 00 76 00 6A 00 4C 00 76 00 63 00 71 00
42. 71 00 38 00 70 00 47 00 69 00 54 00 64 00 51 00
43. 2F 00 76 00 37 00 6F 00 44 00 6B 00 38 00 78 00
44. 5A 00 42 00 6E 00 61 00 73 00 4C 00 36 00 71 00
45. 2F 00 62 00 38 00 36 00 73 00 5A 00 6C 00 2B 00
46. 55 00 35 00 77 00 69 00 4C 00 50 00 50 00 54 00
47. 68 00 67 00 4D 00 70 00 49 00 46 00 2F 00 42 00
48. 61 00 43 00 2F 00 72 00 45 00 4F 00 51 00 50 00
49. 6B 00 6B 00 63 00 51 00 4A 00 4F 00 56 00 54 00
50. 71 00 51 00 72 00 64 00 63 00 4B 00 4F 00 4D 00
51. 56 00 4E 00 4A 00 61 00 55 00 50 00 49 00 52 00
52. 33 00 64 00 2B 00 42 00 62 00 43 00 66 00 31 00
53. 6D 00 5A 00 62 00 7A 00 6F 00 6E 00 64 00 4F 00
54. 51 00 77 00 39 00 57 00 37 00 49 00 77 00 63 00
55. 6D 00 30 00 45 00 44 00 69 00 78 00 51 00 71 00
56. 70 00 41 00 74 00 61 00 75 00 63 00 42 00 48 00
57. 4F 00 33 00 46 00 75 00 37 00 6D 00 49 00 63 00
58. 69 00 78 00 46 00 39 00 67 00 53 00 2F 00 46 00
59. 4B 00 38 00 2F 00 37 00 6C 00 6E 00 78 00 34 00
60. 53 00 6E 00 33 00 5A 00 2F 00 51 00 7A 00 2B 00
61. 48 00 76 00 6A 00 4C 00 49 00 57 00 2F 00 76 00
62. 44 00 77 00 7A 00 77 00 44 00 5A 00 67 00 41 00
63. 41 00 43 00 4E 00 6D 00 73 00 64 00 34 00 65 00
64. 79 00 68 00 4E 00 49 00 75 00 59 00 41 00 47 00
65. 62 00 58 00 62 00 41 00 4B 00 51 00 37 00 63 00
66. 4A 00 66 00 31 00 6B 00 78 00 31 00 31 00 6D 00
67. 4E 00 42 00 49 00 34 00 79 00 42 00 44 00 36 00
68. 4E 00 4E 00 54 00 53 00 6F 00 63 00 75 00 46 00
69. 42 00 6A 00 4F 00 6B 00 2B 00 73 00 41 00 5A 00
70. 4C 00 35 00 5A 00 34 00 56 00 4E 00 32 00 4D 00
71. 4C 00 71 00 49 00 35 00 71 00 38 00 58 00 54 00
72. 58 00 35 00 72 00 58 00 57 00 65 00 79 00 4B 00
73. 75 00 62 00 76 00 49 00 2F 00 59 00 2F 00 6D 00
74. 64 00 42 00 6A 00 64 00 4E 00 36 00 37 00 51 00
75. 57 00 49 00 72 00 75 00 65 00 73 00 4B 00 52 00
76. 56 00 65 00 32 00 35 00 31 00 31 00 54 00 6E 00
77. 42 00 58 00 46 00 62 00 70 00 53 00 47 00 6E 00
78. 42 00 4A 00 65 00 38 00 6F 00 31 00 4B 00 69 00
79. 50 00 55 00 70 00 55 00 4B 00 6A 00 48 00 74 00
80. 54 00 6D 00 75 00 34 00 36 00 4F 00 43 00 4F 00
81. 38 00 4A 00 49 00 62 00 48 00 47 00 6C 00 70 00
82. 4F 00 70 00 62 00 70 00 6E 00 50 00 49 00 73 00
83. 79 00 36 00 39 00 49 00 58 00 6B 00 45 00 7A 00
84. 67 00 38 00 64 00 36 00 34 00 43 00 74 00 65 00
85. 54 00 2F 00 61 00 63 00 73 00 35 00 68 00 4F 00
86. 72 00 62 00 32 00 44 00 73 00 44 00 6F 00 6E 00
87. 4A 00 71 00 65 00 6D 00 46 00 58 00 54 00 47 00
88. 6D 00 35 00 30 00 37 00 30 00 65 00 71 00 35 00
89. 74 00 75 00 6A 00 69 00 5A 00 75 00 43 00 52 00
90. 30 00 4C 00 54 00 36 00 62 00 59 00 49 00 76 00
91. 59 00 64 00 76 00 42 00 6D 00 70 00 52 00 6C 00
92. 31 00 33 00 38 00 41 00 53 00 42 00 32 00 58 00
93. 48 00 30 00 36 00 43 00 4B 00 39 00 33 00 55 00
94. 38 00 63 00 4F 00 54 00 38 00 68 00 68 00 6E 00
95. 6A 00 32 00 53 00 4D 00 49 00 61 00 2B 00 4B 00
96. 30 00 37 00 62 00 72 00 6F 00 66 00 43 00 50 00
97. 4B 00 35 00 37 00 51 00 66 00 43 00 4D 00 35 00
98. 32 00 4F 00 49 00 4B 00 33 00 2F 00 30 00 67 00
99. 70 00 6F 00 4C 00 41 00 6C 00 4A 00 6E 00 49 00
100. 56 00 59 00 72 00 45 00 6B 00 6B 00 41 00 6E 00
101. 47 00 59 00 79 00 70 00 52 00 34 00 45 00 34 00
102. 6F 00 4D 00 4D 00 33 00 63 00 48 00 32 00 38 00
103. 4D 00 42 00 4C 00 66 00 30 00 76 00 47 00 57 00
104. 2F 00 4D 00 62 00 6E 00 74 00 61 00 31 00 35 00
105. 47 00 37 00 47 00 72 00 71 00 76 00 54 00 41 00
106. 73 00 5A 00 30 00 78 00 42 00 6C 00 35 00 76 00
107. 38 00 74 00 44 00 31 00 4F 00 70 00 70 00 6B 00
108. 47 00 6D 00 2B 00 78 00 56 00 62 00 54 00 78 00
109. 49 00 6E 00 2F 00 31 00 7A 00 51 00 67 00 67 00
110. 32 00 71 00 54 00 4F 00 45 00 53 00 6D 00 6E 00
111. 63 00 49 00 4B 00 38 00 49 00 44 00 6A 00 46 00
112. 50 00 7A 00 79 00 59 00 31 00 71 00 38 00 45 00
113. 49 00 6A 00 38 00 39 00 52 00 72 00 2B 00 63 00
114. 6C 00 51 00 4C 00 33 00 62 00 76 00 39 00 69 00
115. 38 00 4F 00 34 00 68 00 77 00 66 00 43 00 62 00
116. 5A 00 51 00 65 00 63 00 38 00 59 00 4F 00 31 00
117. 4D 00 67 00 4F 00 5A 00 32 00 63 00 2B 00 5A 00
118. 74 00 39 00 45 00 55 00 54 00 67 00 31 00 78 00
119. 51 00 49 00 31 00 6A 00 49 00 71 00 76 00 2B 00
120. 65 00 47 00 63 00 62 00 47 00 50 00 66 00 47 00
121. 78 00 4C 00 4C 00 45 00 57 00 76 00 66 00 2B 00
122. 6A 00 54 00 71 00 58 00 63 00 6B 00 55 00 68 00
123. 79 00 4A 00 42 00 47 00 33 00 6D 00 41 00 45 00
124. 3D 00 26 00 70 00 3D 00 DF 31 12 79 58 7C 0F CD
125. 2B ED 31 0D 8A 06 71 A7 C4 A6 DE BB 08 4F 37 12
126. 07 E2 E9 09 C7 B0 2A 1D 74 6C 73 2D 75 6E 69 71
127. 75 65 3A 20 87 E9 38 E3 2C 32 4F 6C F9 8E 5B 0D
128. 40 B2 1B 57 0E AB FE EF 2F CD 58 DA 0F 7F BD 9C
129. D0 8B F2 15 85 F3 83 20 7C 22 F8 17 9A 95 8F FF
130. <PacketHeader>
131. <Type>
132. <BYTE>10 </BYTE>
133. </Type>
134. <Status>
135. <BYTE>01 </BYTE>
136. </Status>
137. <Length>
138. <BYTE>08 </BYTE>
139. <BYTE>10 </BYTE>
140. </Length>
141. <SPID>
142. <BYTE>00 </BYTE>
143. <BYTE>00 </BYTE>
144. </SPID>
145. <Packet>
146. <BYTE>01 </BYTE>
147. </Packet>
148. <Window>
149. <BYTE>00 </BYTE>
150. </Window>
151. </PacketHeader>
152. <PacketData>
153. <Login7>
154. <Length>
155. <DWORD>08 08 00 00 </DWORD>
156. </Length>
157. <TDSVersion>
158. <DWORD>04 00 00 74 </DWORD>
159. </TDSVersion>
160. <PacketSize>
161. <DWORD>00 10 00 00 </DWORD>
162. </PacketSize>
163. <ClientProgVer>
164. <DWORD>00 00 00 07 </DWORD>
165. </ClientProgVer>
166. <ClientPID>
167. <DWORD>40 37 00 00 </DWORD>
168. </ClientPID>
169. <ConnectionID>
170. <DWORD>00 00 00 00 </DWORD>
171. </ConnectionID>
172. <OptionFlags1>
173. <BYTE>E0 </BYTE>
174. </OptionFlags1>
175. <OptionFlags2>
176. <BYTE>03 </BYTE>
177. </OptionFlags2>
178. <TypeFlags>
179. <BYTE>00 </BYTE>
180. </TypeFlags>
181. <OptionFlags3>
182. <BYTE>10 </BYTE>
183. </OptionFlags3>
184. <ClientTimZone>
185. <LONG>E0 01 00 00 </LONG>
186. </ClientTimZone>
187. <ClientLCID>
188. <DWORD>09 04 00 00 </DWORD>
189. </ClientLCID>
190. <OffsetLength>
191. <ibHostName>
192. <USHORT>5E 00 </USHORT>
193. </ibHostName>
194. <cchHostName>
195. <USHORT>0F 00 </USHORT>
196. </cchHostName>
197. <ibUserName>
198. <USHORT>00 00 </USHORT>
199. </ibUserName>
200. <cchUserName>
201. <USHORT>00 00 </USHORT>
202. </cchUserName>
203. <ibPassword>
204. <USHORT>00 00 </USHORT>
205. </ibPassword>
206. <cchPassword>
207. <USHORT>00 00 </USHORT>
208. </cchPassword>
209. <ibAppName>
210. <USHORT>7C 00 </USHORT>
211. </ibAppName>
212. <cchAppName>
213. <USHORT>06 00 </USHORT>
214. </cchAppName>
215. <ibServerName>
216. <USHORT>88 00 </USHORT>
217. </ibServerName>
218. <cchServerName>
219. <USHORT>18 00 </USHORT>
220. </cchServerName>
221. <ibExtension>
222. <USHORT>B8 00 </USHORT>
223. </ibExtension>
224. <cbExtension>
225. <USHORT>04 00 </USHORT>
226. </cbExtension>
227. <ibCltIntName>
228. <USHORT>BC 00 </USHORT>
229. </ibCltIntName>
230. <cchCltIntName>
231. <USHORT>04 00 </USHORT>
232. </cchCltIntName>
233. <ibLanguage>
234. <USHORT>C4 00 </USHORT>
235. </ibLanguage>
236. <cchLanguage>
237. <USHORT>00 00 </USHORT>
238. </cchLanguage>
239. <ibDatabase>
240. <USHORT>C4 00 </USHORT>
241. </ibDatabase>
242. <cchDatabase>
243. <USHORT>00 00 </USHORT>
244. </cchDatabase>
245. <ClientID>
246. <BYTES>00 15 5D 71 E7 42 </BYTES>
247. </ClientID>
248. <ibSSPI>
249. <USHORT>00 00 </USHORT>
250. </ibSSPI>
251. <cbSSPI>
252. <USHORT>00 00 </USHORT>
253. </cbSSPI>
254. <ibAtchDBFile>
255. <USHORT>C4 00 </USHORT>
256. </ibAtchDBFile>
257. <cchAtchDBFile>
258. <USHORT>00 00 </USHORT>
259. </cchAtchDBFile>
260. <ibChangePassword>
261. <USHORT>00 00 </USHORT>
262. </ibChangePassword>
263. <cchChangePassword>
264. <USHORT>00 00 </USHORT>
265. </cchChangePassword>
266. <cbSSPILong>
267. <LONG>00 00 00 00 </LONG>
268. </cbSSPILong>
269. </OffsetLength>
270. <Data>
271. <BYTES>44 00 41 00 4E 00 42 00 45 00
272. 4E 00 45 00 44 00 33 00 2D 00 58 00 47 00 5A 00
273. 55 00 4F 00 53 00 51 00 4C 00 43 00 4D 00 44 00
274. 63 00 6C 00 6F 00 75 00 64 00 2E 00 64 00 65 00
275. 76 00 2E 00 6D 00 73 00 63 00 64 00 73 00 2E 00
276. 63 00 6F 00 6D 00 2C 00 31 00 34 00 33 00 35 00
277. C4 00 00 00 4F 00 44 00 42 00 43 00 </BYTES>
278. </Data>
279. <FeatureExt>
280. <FeatureOpt>
281. <FeatureId>
282. <BYTE>02 </BYTE>
283. </FeatureId>
284. <FeatureDataLen>
285. <DWORD>3E 07 00 00 </DWORD>
286. </FeatureDataLen>
287. <FeatureData>
288. <Options>
289. <BYTE>01 </BYTE>
290. </Options>
291. <FedAuthToken>
292. <L\_VARBYTE>
293. <LONGLEN>
294. <DWORD>E2 06 00 00 </DWORD>
295. </LONGLEN>
296. <BYTES>74 00 3D 00 45 00 77 00 43 00
297. 51 00 41 00 6A 00 4B 00 6A 00 42 00 77 00 41 00
298. 55 00 53 00 30 00 48 00 6F 00 2F 00 30 00 65 00
299. 31 00 52 00 35 00 4D 00 32 00 77 00 37 00 74 00
300. 69 00 44 00 72 00 36 00 70 00 4D 00 63 00 75 00
301. 77 00 33 00 35 00 6B 00 41 00 41 00 61 00 62 00
302. 76 00 45 00 38 00 6F 00 55 00 45 00 71 00 2F 00
303. 50 00 45 00 75 00 4C 00 37 00 32 00 64 00 54 00
304. 74 00 64 00 4D 00 42 00 43 00 7A 00 46 00 37 00
305. 41 00 64 00 4A 00 6D 00 6E 00 70 00 45 00 68 00
306. 34 00 32 00 65 00 41 00 46 00 57 00 37 00 2F 00
307. 38 00 49 00 6A 00 41 00 53 00 6D 00 78 00 79 00
308. 50 00 30 00 30 00 71 00 59 00 64 00 76 00 65 00
309. 31 00 5A 00 6B 00 45 00 2F 00 58 00 6C 00 63 00
310. 4E 00 79 00 69 00 77 00 6F 00 55 00 76 00 54 00
311. 4E 00 4F 00 6B 00 74 00 42 00 45 00 4C 00 71 00
312. 76 00 37 00 58 00 4A 00 54 00 6F 00 47 00 71 00
313. 56 00 32 00 6F 00 6B 00 51 00 6E 00 2F 00 65 00
314. 63 00 50 00 41 00 78 00 32 00 71 00 6A 00 55 00
315. 57 00 74 00 6B 00 54 00 59 00 56 00 66 00 62 00
316. 72 00 51 00 58 00 78 00 76 00 4E 00 58 00 66 00
317. 69 00 72 00 77 00 47 00 57 00 63 00 43 00 45 00
318. 4B 00 79 00 46 00 64 00 76 00 35 00 62 00 78 00
319. 55 00 75 00 68 00 69 00 49 00 62 00 39 00 42 00
320. 55 00 56 00 54 00 56 00 4B 00 6A 00 57 00 51 00
321. 34 00 78 00 4E 00 46 00 4E 00 6B 00 43 00 33 00
322. 73 00 58 00 71 00 6F 00 46 00 52 00 31 00 49 00
323. 46 00 64 00 76 00 6A 00 4C 00 76 00 63 00 71 00
324. 71 00 38 00 70 00 47 00 69 00 54 00 64 00 51 00
325. 2F 00 76 00 37 00 6F 00 44 00 6B 00 38 00 78 00
326. 5A 00 42 00 6E 00 61 00 73 00 4C 00 36 00 71 00
327. 2F 00 62 00 38 00 36 00 73 00 5A 00 6C 00 2B 00
328. 55 00 35 00 77 00 69 00 4C 00 50 00 50 00 54 00
329. 68 00 67 00 4D 00 70 00 49 00 46 00 2F 00 42 00
330. 61 00 43 00 2F 00 72 00 45 00 4F 00 51 00 50 00
331. 6B 00 6B 00 63 00 51 00 4A 00 4F 00 56 00 54 00
332. 71 00 51 00 72 00 64 00 63 00 4B 00 4F 00 4D 00
333. 56 00 4E 00 4A 00 61 00 55 00 50 00 49 00 52 00
334. 33 00 64 00 2B 00 42 00 62 00 43 00 66 00 31 00
335. 6D 00 5A 00 62 00 7A 00 6F 00 6E 00 64 00 4F 00
336. 51 00 77 00 39 00 57 00 37 00 49 00 77 00 63 00
337. 6D 00 30 00 45 00 44 00 69 00 78 00 51 00 71 00
338. 70 00 41 00 74 00 61 00 75 00 63 00 42 00 48 00
339. 4F 00 33 00 46 00 75 00 37 00 6D 00 49 00 63 00
340. 69 00 78 00 46 00 39 00 67 00 53 00 2F 00 46 00
341. 4B 00 38 00 2F 00 37 00 6C 00 6E 00 78 00 34 00
342. 53 00 6E 00 33 00 5A 00 2F 00 51 00 7A 00 2B 00
343. 48 00 76 00 6A 00 4C 00 49 00 57 00 2F 00 76 00
344. 44 00 77 00 7A 00 77 00 44 00 5A 00 67 00 41 00
345. 41 00 43 00 4E 00 6D 00 73 00 64 00 34 00 65 00
346. 79 00 68 00 4E 00 49 00 75 00 59 00 41 00 47 00
347. 62 00 58 00 62 00 41 00 4B 00 51 00 37 00 63 00
348. 4A 00 66 00 31 00 6B 00 78 00 31 00 31 00 6D 00
349. 4E 00 42 00 49 00 34 00 79 00 42 00 44 00 36 00
350. 4E 00 4E 00 54 00 53 00 6F 00 63 00 75 00 46 00
351. 42 00 6A 00 4F 00 6B 00 2B 00 73 00 41 00 5A 00
352. 4C 00 35 00 5A 00 34 00 56 00 4E 00 32 00 4D 00
353. 4C 00 71 00 49 00 35 00 71 00 38 00 58 00 54 00
354. 58 00 35 00 72 00 58 00 57 00 65 00 79 00 4B 00
355. 75 00 62 00 76 00 49 00 2F 00 59 00 2F 00 6D 00
356. 64 00 42 00 6A 00 64 00 4E 00 36 00 37 00 51 00
357. 57 00 49 00 72 00 75 00 65 00 73 00 4B 00 52 00
358. 56 00 65 00 32 00 35 00 31 00 31 00 54 00 6E 00
359. 42 00 58 00 46 00 62 00 70 00 53 00 47 00 6E 00
360. 42 00 4A 00 65 00 38 00 6F 00 31 00 4B 00 69 00
361. 50 00 55 00 70 00 55 00 4B 00 6A 00 48 00 74 00
362. 54 00 6D 00 75 00 34 00 36 00 4F 00 43 00 4F 00
363. 38 00 4A 00 49 00 62 00 48 00 47 00 6C 00 70 00
364. 4F 00 70 00 62 00 70 00 6E 00 50 00 49 00 73 00
365. 79 00 36 00 39 00 49 00 58 00 6B 00 45 00 7A 00
366. 67 00 38 00 64 00 36 00 34 00 43 00 74 00 65 00
367. 54 00 2F 00 61 00 63 00 73 00 35 00 68 00 4F 00
368. 72 00 62 00 32 00 44 00 73 00 44 00 6F 00 6E 00
369. 4A 00 71 00 65 00 6D 00 46 00 58 00 54 00 47 00
370. 6D 00 35 00 30 00 37 00 30 00 65 00 71 00 35 00
371. 74 00 75 00 6A 00 69 00 5A 00 75 00 43 00 52 00
372. 30 00 4C 00 54 00 36 00 62 00 59 00 49 00 76 00
373. 59 00 64 00 76 00 42 00 6D 00 70 00 52 00 6C 00
374. 31 00 33 00 38 00 41 00 53 00 42 00 32 00 58 00
375. 48 00 30 00 36 00 43 00 4B 00 39 00 33 00 55 00
376. 38 00 63 00 4F 00 54 00 38 00 68 00 68 00 6E 00
377. 6A 00 32 00 53 00 4D 00 49 00 61 00 2B 00 4B 00
378. 30 00 37 00 62 00 72 00 6F 00 66 00 43 00 50 00
379. 4B 00 35 00 37 00 51 00 66 00 43 00 4D 00 35 00
380. 32 00 4F 00 49 00 4B 00 33 00 2F 00 30 00 67 00
381. 70 00 6F 00 4C 00 41 00 6C 00 4A 00 6E 00 49 00
382. 56 00 59 00 72 00 45 00 6B 00 6B 00 41 00 6E 00
383. 47 00 59 00 79 00 70 00 52 00 34 00 45 00 34 00
384. 6F 00 4D 00 4D 00 33 00 63 00 48 00 32 00 38 00
385. 4D 00 42 00 4C 00 66 00 30 00 76 00 47 00 57 00
386. 2F 00 4D 00 62 00 6E 00 74 00 61 00 31 00 35 00
387. 47 00 37 00 47 00 72 00 71 00 76 00 54 00 41 00
388. 73 00 5A 00 30 00 78 00 42 00 6C 00 35 00 76 00
389. 38 00 74 00 44 00 31 00 4F 00 70 00 70 00 6B 00
390. 47 00 6D 00 2B 00 78 00 56 00 62 00 54 00 78 00
391. 49 00 6E 00 2F 00 31 00 7A 00 51 00 67 00 67 00
392. 32 00 71 00 54 00 4F 00 45 00 53 00 6D 00 6E 00
393. 63 00 49 00 4B 00 38 00 49 00 44 00 6A 00 46 00
394. 50 00 7A 00 79 00 59 00 31 00 71 00 38 00 45 00
395. 49 00 6A 00 38 00 39 00 52 00 72 00 2B 00 63 00
396. 6C 00 51 00 4C 00 33 00 62 00 76 00 39 00 69 00
397. 38 00 4F 00 34 00 68 00 77 00 66 00 43 00 62 00
398. 5A 00 51 00 65 00 63 00 38 00 59 00 4F 00 31 00
399. 4D 00 67 00 4F 00 5A 00 32 00 63 00 2B 00 5A 00
400. 74 00 39 00 45 00 55 00 54 00 67 00 31 00 78 00
401. 51 00 49 00 31 00 6A 00 49 00 71 00 76 00 2B 00
402. 65 00 47 00 63 00 62 00 47 00 50 00 66 00 47 00
403. 78 00 4C 00 4C 00 45 00 57 00 76 00 66 00 2B 00
404. 6A 00 54 00 71 00 58 00 63 00 6B 00 55 00 68 00
405. 79 00 4A 00 42 00 47 00 33 00 6D 00 41 00 45 00
406. 3D 00 26 00 70 00 3D 00 </BYTES>
407. </L\_VARBYTE>
408. </FedAuthToken>
409. <SignedData>
410. <Nonce>
411. <BYTES>DF 31 12 79 58 7C 0F CD
412. 2B ED 31 0D 8A 06 71 A7 C4 A6 DE BB 08 4F 37 12
413. 07 E2 E9 09 C7 B0 2A 1D </BYTES>
414. </Nonce>
415. <ChannelBindingToken>
416. <BYTES>74 6C 73 2D 75 6E 69 71
417. 75 65 3A 20 87 E9 38 E3 2C 32 4F 6C F9 8E 5B </BYTES>
418. </ChannelBindingToken>
419. <Signature>
420. <BYTES>0D
421. 40 B2 1B 57 0E AB FE EF 2F CD 58 DA 0F 7F BD 9C
422. D0 8B F2 15 85 F3 83 20 7C 22 F8 17 9A 95 8F </BYTES>
423. </Signature>
424. </SignedData>
425. </FeatureData>
426. </FeatureOpt>
427. <TERMINATOR>
428. <BYTE>FF </BYTE>
429. </TERMINATOR>
430. </FeatureExt>
431. </Login7>
432. </PacketData>

## Login Response

Login response from the server to the client:

1. 04 01 01 61 00 00 01 00 E3 1B 00 01 06 6D 00 61
2. 00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
3. 74 00 65 00 72 00 AB 58 00 45 16 00 00 02 00 25
4. 00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
5. 00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
6. 00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
7. 00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
8. 00 74 00 65 00 72 00 27 00 2E 00 00 00 00 00 00
9. 00 E3 08 00 07 05 09 04 D0 00 34 00 E3 17 00 02
10. 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69
11. 00 73 00 68 00 00 E3 13 00 04 04 34 00 30 00 39
12. 00 36 00 04 34 00 30 00 39 00 36 00 AB 5C 00 47
13. 16 00 00 01 00 27 00 43 00 68 00 61 00 6E 00 67
14. 00 65 00 64 00 20 00 6C 00 61 00 6E 00 67 00 75
15. 00 61 00 67 00 65 00 20 00 73 00 65 00 74 00 74
16. 00 69 00 6E 00 67 00 20 00 74 00 6F 00 20 00 75
17. 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73
18. 00 68 00 2E 00 00 00 00 00 00 00 AD 36 00 01 72
19. 09 00 02 16 4D 00 69 00 63 00 72 00 6F 00 73 00
20. 6F 00 66 00 74 00 20 00 53 00 51 00 4C 00 20 00
21. 53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00
22. 00 00 00 00 FD 00 00 00 00 00 00 00 00 00 00 00
23. 00
24. <PacketHeader>
25. <Type>
26. <BYTE>04 </BYTE>
27. </Type>
28. <Status>
29. <BYTE>01 </BYTE>
30. </Status>
31. <Length>
32. <BYTE>01 </BYTE>
33. <BYTE>61 </BYTE>
34. </Length>
35. <SPID>
36. <BYTE>00 </BYTE>
37. <BYTE>00 </BYTE>
38. </SPID>
39. <Packet>
40. <BYTE>01 </BYTE>
41. </Packet>
42. <Window>
43. <BYTE>00 </BYTE>
44. </Window>
45. </PacketHeader>
46. <PacketData>
47. <TableResponse>
48. <ENVCHANGE>
49. <TokenType>
50. <BYTE>E3 </BYTE>
51. </TokenType>
52. <Length>
53. <USHORT>1B 00 </USHORT>
54. </Length>
55. <EnvChangeData>
56. <BYTES>01 06 6D 00 61 00 73 00 74 00 65 00 72 00 06 6D 00 61 00
57. 73 00 74 00 65 00 72 00 </BYTES>
58. </EnvChangeData>
59. </ENVCHANGE>
60. <INFO>
61. <TokenType>
62. <BYTE>AB </BYTE>
63. </TokenType>
64. <Length>
65. <USHORT>58 00 </USHORT>
66. </Length>
67. <Number>
68. <LONG>45 16 00 00 </LONG>
69. </Number>
70. <State>
71. <BYTE>02 </BYTE>
72. </State>
73. <Class>
74. <BYTE>00 </BYTE>
75. </Class>
76. <MsgText>
77. <US\_UNICODE>
78. <USHORTLEN>
79. <USHORT>25 00 </USHORT>
80. </USHORTLEN>
81. <BYTES ascii="C.h.a.n.g.e.d. .d.a.t.a.b.a.s.e. .c.o.n.t.e.x.t.
82. .t.o. .'.m.a.s.t.e.r.'...">43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
83. 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
84. 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
85. 65 00 72 00 27 00 2E 00 </BYTES>
86. </US\_UNICODE>
87. </MsgText>
88. <ServerName>
89. <B\_UNICODE>
90. <BYTELEN>
91. <BYTE>00 </BYTE>
92. </BYTELEN>
93. <BYTES ascii="">
94. </BYTES>
95. </B\_UNICODE>
96. </ServerName>
97. <ProcName>
98. <B\_UNICODE>
99. <BYTELEN>
100. <BYTE>00 </BYTE>
101. </BYTELEN>
102. <BYTES ascii="">
103. </BYTES>
104. </B\_UNICODE>
105. </ProcName>
106. <LineNumber>
107. <LONG>00 00 00 00 </LONG>
108. </LineNumber>
109. </INFO>
110. <ENVCHANGE>
111. <TokenType>
112. <BYTE>E3 </BYTE>
113. </TokenType>
114. <Length>
115. <USHORT>08 00 </USHORT>
116. </Length>
117. <EnvChangeData>
118. <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
119. </EnvChangeData>
120. </ENVCHANGE>
121. <ENVCHANGE>
122. <TokenType>
123. <BYTE>E3 </BYTE>
124. </TokenType>
125. <Length>
126. <USHORT>17 00 </USHORT>
127. </Length>
128. <EnvChangeData>
129. <BYTES>02 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73
130. 00 68 00 00 </BYTES>
131. </EnvChangeData>
132. </ENVCHANGE>
133. <INFO>
134. <TokenType>
135. <BYTE>AB </BYTE>
136. </TokenType>
137. <Length>
138. <USHORT>5C 00 </USHORT>
139. </Length>
140. <Number>
141. <LONG>47 16 00 00 </LONG>
142. </Number>
143. <State>
144. <BYTE>01 </BYTE>
145. </State>
146. <Class>
147. <BYTE>00 </BYTE>
148. </Class>
149. <MsgText>
150. <US\_UNICODE>
151. <USHORTLEN>
152. <USHORT>27 00 </USHORT>
153. </USHORTLEN>
154. <BYTES ascii="C.h.a.n.g.e.d. .l.a.n.g.u.a.g.e. .s.e.t.t.i.n.g.
155. .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
156. 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
157. 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
158. 00 67 00 6C 00 69 00 73 00 68 00 2E 00 </BYTES>
159. </US\_UNICODE>
160. </MsgText>
161. <ServerName>
162. <B\_UNICODE>
163. <BYTELEN>
164. <BYTE>00 </BYTE>
165. </BYTELEN>
166. <BYTES ascii="">
167. </BYTES>
168. </B\_UNICODE>
169. </ServerName>
170. <ProcName>
171. <B\_UNICODE>
172. <BYTELEN>
173. <BYTE>00 </BYTE>
174. </BYTELEN>
175. <BYTES ascii="">
176. </BYTES>
177. </B\_UNICODE>
178. </ProcName>
179. <LineNumber>
180. <LONG>00 00 00 00 </LONG>
181. </LineNumber>
182. </INFO>
183. <LOGINACK>
184. <TokenType>
185. <BYTE>AD </BYTE>
186. </TokenType>
187. <Length>
188. <USHORT>36 00 </USHORT>
189. </Length>
190. <Interface>
191. <BYTE>01 </BYTE>
192. </Interface>
193. <TDSVersion>
194. <DWORD>72 09 00 02 </DWORD>
195. </TDSVersion>
196. <ProgName>
197. <B\_UNICODE>
198. <BYTELEN>
199. <BYTE>16 </BYTE>
200. </BYTELEN>
201. <BYTES ascii="M.i.c.r.o.s.o.f.t. .S.Q.L. .S.e.r.v.e.r.....">4D
202. 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
203. 20 00 53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00 </BYTES>
204. </B\_UNICODE>
205. </ProgName>
206. <ProgVersion>
207. <DWORD>00 00 00 00 </DWORD>
208. </ProgVersion>
209. </LOGINACK>
210. <ENVCHANGE>
211. <TokenType>
212. <BYTE>E3 </BYTE>
213. </TokenType>
214. <Length>
215. <USHORT>13 00 </USHORT>
216. </Length>
217. <EnvChangeData>
218. <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00
219. </BYTES>
220. </EnvChangeData>
221. </ENVCHANGE>
222. <DONE>
223. <TokenType>
224. <BYTE>FD </BYTE>
225. </TokenType>
226. <Status>
227. <USHORT>00 00 </USHORT>
228. </Status>
229. <CurCmd>
230. <USHORT>00 00 </USHORT>
231. </CurCmd>
232. <DoneRowCount>
233. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
234. </DoneRowCount>
235. </DONE>
236. </TableResponse>
237. </PacketData>

## Login Response with Federated Authentication Feature Extension Acknowledgement

Login response from server to client, including the Feature Extension Acknowledgement token that contains a federated authentication feature extension option:

1. 04 01 01 BC 01 4A 01 00 E3 1B 00 01 06 6D 00 61
2. 00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
3. 74 00 65 00 72 00 AB 62 00 45 16 00 00 02 00 25
4. 00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
5. 00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
6. 00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
7. 00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
8. 00 74 00 65 00 72 00 27 00 2E 00 05 63 00 6C 00
9. 6F 00 75 00 64 00 00 01 00 00 00 E3 08 00 07 05
10. 09 04 D0 00 34 00 E3 17 00 02 0A 75 00 73 00 5F
11. 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00
12. AB 66 00 47 16 00 00 01 00 27 00 43 00 68 00 61
13. 00 6E 00 67 00 65 00 64 00 20 00 6C 00 61 00 6E
14. 00 67 00 75 00 61 00 67 00 65 00 20 00 73 00 65
15. 00 74 00 74 00 69 00 6E 00 67 00 20 00 74 00 6F
16. 00 20 00 75 00 73 00 5F 00 65 00 6E 00 67 00 6C
17. 00 69 00 73 00 68 00 2E 00 05 63 00 6C 00 6F 00
18. 75 00 64 00 00 01 00 00 00 AD 36 00 01 74 00 00
19. 04 16 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00
20. 66 00 74 00 20 00 53 00 51 00 4C 00 20 00 53 00
21. 65 00 72 00 76 00 65 00 72 00 00 00 00 00 0B 00
22. 08 CB E3 13 00 04 04 34 00 30 00 39 00 36 00 04
23. 34 00 30 00 39 00 36 00 AE 02 40 00 00 00 C9 08
24. 46 4E 58 49 0C 71 80 72 CD 69 F0 EC 3D E2 F6 ED
25. 75 8C 77 7D 9C B8 BB 87 4A 9C 90 80 A4 EE 40 B6
26. 07 71 0E A8 3C 0E D7 DE 14 DE F3 8B 65 C2 06 8C
27. F9 51 D3 BC 32 55 15 A5 E4 A2 45 62 78 80 FF FD
28. 00 00 00 00 00 00 00 00 00 00 00 00
29. <PacketHeader>
30. <Type>
31. <BYTE>04 </BYTE>
32. </Type>
33. <Status>
34. <BYTE>01 </BYTE>
35. </Status>
36. <Length>
37. <BYTE>01 </BYTE>
38. <BYTE>BC </BYTE>
39. </Length>
40. <SPID>
41. <BYTE>01 </BYTE>
42. <BYTE>4A </BYTE>
43. </SPID>
44. <Packet>
45. <BYTE>01 </BYTE>
46. </Packet>
47. <Window>
48. <BYTE>00 </BYTE>
49. </Window>
50. </PacketHeader>
51. <PacketData>
52. <TableResponse>
53. <ENVCHANGE>
54. <TokenType>
55. <BYTE>E3 </BYTE>
56. </TokenType>
57. <Length>
58. <USHORT>1B 00 </USHORT>
59. </Length>
60. <EnvChangeData>
61. <BYTES>01 06 6D 00 61 00 73 00 74 00 65 00
62. 72 00 06 6D 00 61 00 73 00 74 00 65 00 72 00 </BYTES>
63. </EnvChangeData>
64. </ENVCHANGE>
65. <INFO>
66. <TokenType>
67. <BYTE>AB </BYTE>
68. </TokenType>
69. <Length>
70. <USHORT>62 00 </USHORT>
71. </Length>
72. <Number>
73. <LONG>45 16 00 00 </LONG>
74. </Number>
75. <State>
76. <BYTE>02 </BYTE>
77. </State>
78. <Class>
79. <BYTE>00 </BYTE>
80. </Class>
81. <MsgText>
82. <US\_UNICODE>
83. <USHORTLEN>
84. <USHORT>25 00 </USHORT>
85. </USHORTLEN>
86. <BYTES ascii="C.h.a.n.g.e.d. .d.a.t.a.b.a.s.e. .c.o.n.t.e.x.t.
87. .t.o. .'.m.a.s.t.e.r.'...">
88. 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
89. 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65 00
90. 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74 00
91. 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73 00
92. 74 00 65 00 72 00 27 00 2E 00 </BYTES>
93. </US\_UNICODE>
94. </MsgText>
95. <ServerName>
96. <B\_UNICODE>
97. <BYTELEN>
98. <BYTE>05 </BYTE>
99. </BYTELEN>
100. <BYTES ascii="c.l.o.u.d.">
101. 63 00 6C 00 6F 00 75 00 64 00 </BYTES>
102. </B\_UNICODE>
103. </ServerName>
104. <ProcName>
105. <B\_UNICODE>
106. <BYTELEN>
107. <BYTE>00 </BYTE>
108. </BYTELEN>
109. <BYTES ascii="">
110. </BYTES>
111. </B\_UNICODE>
112. </ProcName>
113. <LineNumber>
114. <LONG>01 00 00 00
115. </LineNumber>
116. </INFO>
117. <ENVCHANGE>
118. <TokenType>
119. <BYTE>E3 </BYTE>
120. </TokenType>
121. <Length>
122. <USHORT>08 00 </USHORT>
123. </Length>
124. <EnvChangeData>
125. <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
126. </EnvChangeData>
127. </ENVCHANGE>
128. <ENVCHANGE>
129. <TokenType>
130. <BYTE>E3 </BYTE>
131. </TokenType>
132. <Length>
133. <USHORT>17 00 </USHORT>
134. </Length>
135. <EnvChangeData>
136. <BYTES>02 0A 75 00 73 00 5F
137. 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00 </BYTES>
138. </EnvChangeData>
139. </ENVCHANGE>
140. <INFO>
141. <TokenType>
142. <BYTE>AB </BYTE>
143. </TokenType>
144. <Length>
145. <USHORT>66 00 </USHORT>
146. </Length>
147. <Number>
148. <LONG>47 16 00 00 </LONG>
149. </Number>
150. <State>
151. <BYTE>01 </BYTE>
152. </State>
153. <Class>
154. <BYTE>00 </BYTE>
155. </Class>
156. <MsgText>
157. <US\_UNICODE>
158. <USHORTLEN>
159. <USHORT>27 00 </USHORT>
160. </USHORTLEN>
161. <BYTES ascii="C.h.a.n.g.e.d. .l.a.n.g.u.a.g.e. .s.e.t.t.i.n.g.
162. .t.o. .u.s.\_.e.n.g.l.i.s.h...">
163. 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20 00
164. 6C 00 61 00 6E 00 67 00 75 00 61 00 67 00 65 00
165. 20 00 73 00 65 00 74 00 74 00 69 00 6E 00 67 00
166. 20 00 74 00 6F 00 20 00 75 00 73 00 5F 00 65 00
167. 6E 00 67 00 6C 00 69 00 73 00 68 00 2E 00 </BYTES>
168. </US\_UNICODE>
169. </MsgText>
170. <ServerName>
171. <B\_UNICODE>
172. <BYTELEN>
173. <BYTE>05 </Byte>
174. </BYTELEN>
175. <BYTES ascii="c.l.o.u.d.">
176. 63 00 6C 00 6F 00 75 00 64 00 </BYTES>
177. </B\_UNICODE>
178. </ServerName>
179. <ProcName>
180. <B\_UNICODE>
181. <BYTELEN>
182. <BYTE>00 </BYTE>
183. </BYTELEN>
184. <BYTES ascii="">
185. </BYTES>
186. </B\_UNICODE>
187. </ProcName>
188. <LineNumber>
189. <LONG>01 00 00 00 </LONG>
190. </LineNumber>
191. </INFO>
192. <LOGINACK>
193. <TokenType>
194. <BYTE>AD </BYTE>
195. </TokenType>
196. <Length>
197. <USHORT>36 00 </USHORT>
198. </Length>
199. <Interface>
200. <BYTE>01 </BYTE>
201. </Interface>
202. <TDSVersion>
203. <DWORD>74 00 00 04 </DWORD>
204. </TDSVersion>
205. <ProgName>
206. <B\_UNICODE>
207. <BYTELEN>
208. <BYTE>16 </BYTE>
209. </BYTELEN>
210. <BYTES ascii="M.i.c.r.o.s.o.f.t. .S.Q.L. .S.e.r.v.e.r.....">4D
211. 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
212. 20 00 53 00 65 00 72 00 76 00 65 00 72 00 00 00 00 00 </BYTES>
213. </B\_UNICODE>
214. </ProgName>
215. <ProgVersion>
216. <DWORD>0B 00 08 CB </DWORD>
217. </ProgVersion>
218. </LOGINACK>
219. <ENVCHANGE>
220. <TokenType>
221. <BYTE>E3 </BYTE>
222. </TokenType>
223. <Length>
224. <USHORT>13 00 </USHORT>
225. </Length>
226. <EnvChangeData>
227. <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00
228. </BYTES>
229. </EnvChangeData>
230. </ENVCHANGE>
231. <FEATUREEXTACK>
232. <TokenType>
233. <BYTE>AE </BYTE>
234. </TokenType>
235. <FeatureAckOpt>
236. <FeatureId>
237. <BYTE>02 </BYTE>
238. </FeatureId>
239. <FeatureAckDataLen>
240. <DWORD>40 00 00 00 </DWORD>
241. </FeatureAckDataLen>
242. <FeatureAckData>
243. <Nonce>
244. <BYTES>
245. C9 08 46 4E 58 49 0C 71 80 72 CD 69 F0 EC 3D E2
246. F6 ED 75 8C 77 7D 9C B8 BB 87 4A 9C 90 80 A4 EE </BYTES>
247. </Nonce>
248. <Signature>
249. <BYTES>
250. 40 B6 07 71 0E A8 3C 0E D7 DE 14 DE F3 8B 65 C2
251. 06 8C F9 51 D3 BC 32 55 15 A5 E4 A2 45 62 78 80 </BYTES>
252. </Signature>
253. </FeatureAckData>
254. </FeatureAckOpt>
255. <FeatureAckOpt>
256. <TERMINATOR>
257. <BYTE>FF </BYTE>
258. </TERMINATOR>
259. </FeatureAckOpt>
260. </FEATUREEXTACK>
261. <DONE>
262. <TokenType>
263. <BYTE>FD </BYTE>
264. </TokenType>
265. <Status>
266. <USHORT>00 00 </USHORT>
267. </Status>
268. <CurCmd>
269. <USHORT>00 00 </USHORT>
270. </CurCmd>
271. <DoneRowCount>
272. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
273. </DoneRowCount>
274. </DONE>
275. </TableResponse>
276. </PacketData>

## SQL Batch Client Request

Client request sent from the client to the server:

1. 01 01 00 5C 00 00 01 00 16 00 00 00 12 00 00 00
2. 02 00 00 00 00 00 00 00 00 01 00 00 00 00 0A 00
3. 73 00 65 00 6C 00 65 00 63 00 74 00 20 00 27 00
4. 66 00 6F 00 6F 00 27 00 20 00 61 00 73 00 20 00
5. 27 00 62 00 61 00 72 00 27 00 0A 00 20 00 20 00
6. 20 00 20 00 20 00 20 00 20 00 20 00
7. <PacketHeader>
8. <Type>
9. <BYTE>01 </BYTE>
10. </Type>
11. <Status>
12. <BYTE>01 </BYTE>
13. </Status>
14. <Length>
15. <BYTE>00 </BYTE>
16. <BYTE>5C </BYTE>
17. </Length>
18. <SPID>
19. <BYTE>00 </BYTE>
20. <BYTE>00 </BYTE>
21. </SPID>
22. <Packet>
23. <BYTE>01 </BYTE>
24. </Packet>
25. <Window>
26. <BYTE>00 </BYTE>
27. </Window>
28. </PacketHeader>
29. <PacketData>
30. <SQLBatch>
31. <All\_HEADERS>
32. <TotalLength>
33. <DWORD>16 00 00 00 </DWORD>
34. </TotalLength>
35. <Header>
36. <HeaderLength>
37. <DWORD>12 00 00 00 </DWORD>
38. </HeaderLength>
39. <HeaderType>
40. <USHORT>02 00 </USHORT>
41. </HeaderType>
42. <HeaderData>
43. <MARS>
44. <TransactionDescriptor>
45. <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
46. </TransactionDescriptor>
47. <OutstandingRequestCount>
48. <DWORD>00 00 00 00 </DWORD>
49. </OutstandingRequestCount>
50. </MARS>
51. </HeaderData>
52. </Header>
53. </All\_HEADERS>
54. <SQLText>
55. <UNICODESTREAM>
56. <BYTES>0A 00 73 00 65 00 6C 00 65 00 63 00 74 00 20 00 27 00 66
57. 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
58. 0A 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00 </BYTES>
59. </UNICODESTREAM>
60. </SQLText>
61. </SQLBatch>
62. </PacketData>

## SQL Batch Server Response

Server response sent from the server to the client:

1. 04 01 00 33 00 00 01 00 81 01 00 00 00 00 00 20
2. 00 A7 03 00 09 04 D0 00 34 03 62 00 61 00 72 00
3. D1 03 00 66 6F 6F FD 10 00 C1 00 01 00 00 00 00
4. 00 00 00
5. <PacketHeader>
6. <Type>
7. <BYTE>04 </BYTE>
8. </Type>
9. <Status>
10. <BYTE>01 </BYTE>
11. </Status>
12. <Length>
13. <BYTE>00 </BYTE>
14. <BYTE>33 </BYTE>
15. </Length>
16. <SPID>
17. <BYTE>00 </BYTE>
18. <BYTE>00 </BYTE>
19. </SPID>
20. <Packet>
21. <BYTE>01 </BYTE>
22. </Packet>
23. <Window>
24. <BYTE>00 </BYTE>
25. </Window>
26. </PacketHeader>
27. <PacketData>
28. <TableResponse>
29. <COLMETADATA>
30. <TokenType>
31. <BYTE>81 </BYTE>
32. </TokenType>
33. <Count>
34. <USHORT>01 00 </USHORT>
35. </Count>
36. <ColumnData>
37. <UserType>
38. <ULONG>00 00 00 00 </ULONG>
39. </UserType>
40. <Flags>
41. <USHORT>20 00 </USHORT>
42. </Flags>
43. <TYPE\_INFO>
44. <VARLENTYPE>
45. <USHORTLEN\_TYPE>
46. <BYTE>A7 </BYTE>
47. </USHORTLEN\_TYPE>
48. </VARLENTYPE>
49. <TYPE\_VARLEN>
50. <USHORTCHARBINLEN>
51. <USHORT>03 00 </USHORT>
52. </USHORTCHARBINLEN>
53. </TYPE\_VARLEN>
54. <COLLATION>
55. <BYTES>09 04 D0 00 34 </BYTES>
56. </COLLATION>
57. </TYPE\_INFO>
58. <ColName>
59. <B\_UNICODE>
60. <BYTELEN>
61. <BYTE>03 </BYTE>
62. </BYTELEN>
63. <BYTES ascii="b.a.r.">62 00 61 00 72 00 </BYTES>
64. </B\_UNICODE>
65. </ColName>
66. </ColumnData>
67. </COLMETADATA>
68. <ROW>
69. <TokenType>
70. <BYTE>D1 </BYTE>
71. </TokenType>
72. <TYPE\_VARBYTE>
73. <TYPE\_VARLEN>
74. <USHORTCHARBINLEN>
75. <USHORT>03 00 </USHORT>
76. </USHORTCHARBINLEN>
77. </TYPE\_VARLEN>
78. <BYTES ascii="fio">66 6F 6F </BYTES>
79. </TYPE\_VARBYTE>
80. </ROW>
81. <DONE>
82. <TokenType>
83. <BYTE>FD </BYTE>
84. </TokenType>
85. <Status>
86. <USHORT>10 00 </USHORT>
87. </Status>
88. <CurCmd>
89. <USHORT>C1 00 </USHORT>
90. </CurCmd>
91. <DoneRowCount>
92. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
93. </DoneRowCount>
94. </DONE>
95. </TableResponse>
96. </PacketData>

## RPC Client Request

RPC request sent from the client to the server:

1. 03 01 00 2F 00 00 01 00 16 00 00 00 12 00 00 00
2. 02 00 00 00 00 00 00 00 00 01 00 00 00 00 04 00
3. 66 00 6F 00 6F 00 33 00 00 00 00 02 26 02 00
4. <PacketHeader>
5. <Type>
6. <BYTE>03 </BYTE>
7. </Type>
8. <Status>
9. <BYTE>01 </BYTE>
10. </Status>
11. <Length>
12. <BYTE>00 </BYTE>
13. <BYTE>2F </BYTE>
14. </Length>
15. <SPID>
16. <BYTE>00 </BYTE>
17. <BYTE>00 </BYTE>
18. </SPID>
19. <Packet>
20. <BYTE>01 </BYTE>
21. </Packet>
22. <Window>
23. <BYTE>00 </BYTE>
24. </Window>
25. </PacketHeader>
26. <PacketData>
27. <RPCRequest>
28. <All\_HEADERS>
29. <TotalLength>
30. <DWORD>16 00 00 00 </DWORD>
31. </TotalLength>
32. <Header>
33. <HeaderLength>
34. <DWORD>12 00 00 00 </DWORD>
35. </HeaderLength>
36. <HeaderType>
37. <USHORT>02 00 </USHORT>
38. </HeaderType>
39. <HeaderData>
40. <MARS>
41. <TransactionDescriptor>
42. <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
43. </TransactionDescriptor>
44. <OutstandingRequestCount>
45. <DWORD>00 00 00 00 </DWORD>
46. </OutstandingRequestCount>
47. </MARS>
48. </HeaderData>
49. </Header>
50. </All\_HEADERS>
51. <RPCReqBatch>
52. <NameLenProcID>
53. <ProcName>
54. <US\_UNICODE>
55. <USHORTLEN>
56. <USHORT>04 00 </USHORT>
57. </USHORTLEN>
58. <BYTES ascii="f.o.o.3.">66 00 6F 00 6F 00 33 00 </BYTES>
59. </US\_UNICODE>
60. </ProcName>
61. </NameLenProcID>
62. <OptionFlags>
63. <fWithRecomp>
64. <BIT>0</BIT>
65. </fWithRecomp>
66. <fNoMetaData>
67. <BIT>0</BIT>
68. </fNoMetaData>
69. <fReuseMetaData>
70. <BIT>false</BIT>
71. </fReuseMetaData>
72. </OptionFlags>
73. <ParameterData>
74. <ParamMetaData>
75. <B\_UNICODE>
76. <BYTELEN>
77. <BYTE>00 </BYTE>
78. </BYTELEN>
79. <BYTES ascii="">
80. </BYTES>
81. </B\_UNICODE>
82. <StatusFlags>
83. <fByRefValue>
84. <BIT>0</BIT>
85. </fByRefValue>
86. <fDefaultValue>
87. <BIT>1</BIT>
88. </fDefaultValue>
89. </StatusFlags>
90. <TYPE\_INFO>
91. <VARLENTYPE>
92. <BYTELEN\_TYPE>
93. <BYTE>26 </BYTE>
94. </BYTELEN\_TYPE>
95. </VARLENTYPE>
96. <TYPE\_VARLEN>
97. <BYTELEN>
98. <BYTE>02 </BYTE>
99. </BYTELEN>
100. </TYPE\_VARLEN>
101. </TYPE\_INFO>
102. </ParamMetaData>
103. <ParamLenData>
104. <TYPE\_VARBYTE>
105. <TYPE\_VARLEN>
106. <BYTELEN>
107. <BYTE>00 </BYTE>
108. </BYTELEN>
109. </TYPE\_VARLEN>
110. <BYTES>
111. </BYTES>
112. </TYPE\_VARBYTE>
113. </ParamLenData>
114. </ParameterData>
115. </RPCReqBatch>
116. </RPCRequest>
117. </PacketData>

## RPC Server Response

RPC response sent from the server to the client:

1. 04 01 00 27 00 00 01 00 FF 11 00 C1 00 01 00 00
2. 00 00 00 00 00 79 00 00 00 00 FE 00 00 E0 00 00
3. 00 00 00 00 00 00 00
4. <PacketHeader>
5. <Type>
6. <BYTE>04 </BYTE>
7. </Type>
8. <Status>
9. <BYTE>01 </BYTE>
10. </Status>
11. <Length>
12. <BYTE>00 </BYTE>
13. <BYTE>27 </BYTE>
14. </Length>
15. <SPID>
16. <BYTE>00 </BYTE>
17. <BYTE>00 </BYTE>
18. </SPID>
19. <Packet>
20. <BYTE>01 </BYTE>
21. </Packet>
22. <Window>
23. <BYTE>00 </BYTE>
24. </Window>
25. </PacketHeader>
26. <PacketData>
27. <TableResponse>
28. <DONEINPROC>
29. <TokenType>
30. <BYTE>FF </BYTE>
31. </TokenType>
32. <Status>
33. <USHORT>11 00 </USHORT>
34. </Status>
35. <CurCmd>
36. <USHORT>C1 00 </USHORT>
37. </CurCmd>
38. <DoneRowCount>
39. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
40. </DoneRowCount>
41. </DONEINPROC>
42. <RETURNSTATUS>
43. <TokenType>
44. <BYTE>79 </BYTE>
45. </TokenType>
46. <VALUE>
47. <LONG>00 00 00 00 </LONG>
48. </VALUE>
49. </RETURNSTATUS>
50. <DONEPROC>
51. <TokenType>
52. <BYTE>FE </BYTE>
53. </TokenType>
54. <Status>
55. <USHORT>00 00 </USHORT>
56. </Status>
57. <CurCmd>
58. <USHORT>E0 00 </USHORT>
59. </CurCmd>
60. <DoneRowCount>
61. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
62. </DoneRowCount>
63. </DONEPROC>
64. </TableResponse>
65. </PacketData>

## Attention Request

Attention request sent from client to server:

1. 06 01 00 08 00 00 01 00
2. <PacketHeader>
3. <Type>
4. <BYTE>06</BYTE>
5. </Type>
6. <Status>
7. <BYTE>01</BYTE>
8. </Status>
9. <Length>
10. <BYTE>00</BYTE>
11. <BYTE>08</BYTE>
12. </Length>
13. <SPID>
14. <BYTE>00</BYTE>
15. <BYTE>00</BYTE>
16. </SPID>
17. <Packet>
18. <BYTE>01</BYTE>
19. </Packet>
20. <Window>
21. <BYTE>00</BYTE>
22. </Window>
23. </PacketHeader>

## SSPI Message

SSPI message carrying SSPI payload sent from client to server:

1. 11 01 00 60 00 00 01 00 4E 54 4C 4D 53 53 50 00
2. 03 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
3. 58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
4. 58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00
5. 58 00 00 00 15 C2 88 E2 06 00 71 17 00 00 00 0F
6. 30 81 C1 7D 59 5F E9 3E 1A 7C 98 05 01 72 5C 4F
7. <PacketHeader>
8. <Type>
9. <BYTE>11 </BYTE>
10. </Type>
11. <Status>
12. <BYTE>01 </BYTE>
13. </Status>
14. <Length>
15. <BYTE>00 </BYTE>
16. <BYTE>60 </BYTE>
17. </Length>
18. <SPID>
19. <BYTE>00 </BYTE>
20. <BYTE>00 </BYTE>
21. </SPID>
22. <Packet>
23. <BYTE>01 </BYTE>
24. </Packet>
25. <Window>
26. <BYTE>00 </BYTE>
27. </Window>
28. </PacketHeader>
29. <PacketData>
30. <SSPI>
31. <BYTES>4E 54 4C 4D 53 53 50 00 03 00 00 00 00 00 00 00 58 00 00 00 00
32. 00 00 00 58 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00 58 00 00 00 00
33. 00 00 58 00 00 00 00 00 00 00 58 00 00 00 15 C2 88 E2 06 00 71 17 00 00
34. 0F 30 81 C1 7D 59 5F E9 3E 1A 7C 98 05 01 72 5C 4F </BYTES>
35. </SSPI>
36. </PacketData>

## Bulk Load

BULKLOADBCP request sent from client to server:

1. 07 01 00 26 00 00 01 00 81 01 00 00 00 00 00 05
2. 00 32 02 63 00 31 00 D1 00 FD 00 00 00 00 00 00
3. 00 00 00 00 00 00
4. <PacketHeader>
5. <Type>
6. <BYTE>07 </BYTE>
7. </Type>
8. <Status>
9. <BYTE>01 </BYTE>
10. </Status>
11. <Length>
12. <BYTE>00 </BYTE>
13. <BYTE>26 </BYTE>
14. </Length>
15. <SPID>
16. <BYTE>00 </BYTE>
17. <BYTE>00 </BYTE>
18. </SPID>
19. <Packet>
20. <BYTE>01 </BYTE>
21. </Packet>
22. <Window>
23. <BYTE>00 </BYTE>
24. </Window>
25. </PacketHeader>
26. <PacketData>
27. <BulkLoadBCP>
28. <COLMETADATA>
29. <TokenType>
30. <BYTE>81 </BYTE>
31. </TokenType>
32. <Count>
33. <USHORT>01 00 </USHORT>
34. </Count>
35. <ColumnData>
36. <UserType>
37. <ULONG>00 00 00 00 </ULONG>
38. </UserType>
39. <Flags>
40. <USHORT>05 00 </USHORT>
41. </Flags>
42. <TYPE\_INFO>
43. <FIXEDLENTYPE>
44. <BYTE>32 </BYTE>
45. </FIXEDLENTYPE>
46. </TYPE\_INFO>
47. <ColName>
48. <B\_UNICODE>
49. <BYTELEN>
50. <BYTE>02 </BYTE>
51. </BYTELEN>
52. <BYTES ascii="c.1.">63 00 31 00 </BYTES>
53. </B\_UNICODE>
54. </ColName>
55. </ColumnData>
56. </COLMETADATA>
57. <ROW>
58. <TokenType>
59. <BYTE>D1 </BYTE>
60. </TokenType>
61. <TYPE\_VARBYTE>
62. <BYTES>00 </BYTES>
63. </TYPE\_VARBYTE>
64. </ROW>
65. <DONE>
66. <TokenType>
67. <BYTE>FD </BYTE>
68. </TokenType>
69. <Status>
70. <USHORT>00 00 </USHORT>
71. </Status>
72. <CurCmd>
73. <USHORT>00 00 </USHORT>
74. </CurCmd>
75. <DoneRowCount>
76. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
77. </DoneRowCount>
78. </DONE>
79. </BulkLoadBCP>
80. </PacketData>

## Transaction Manager Request

Transaction Manager Request sent from client to server:

1. 0E 01 00 20 00 00 01 00 16 00 00 00
2. 12 00 00 00 02 00 00 00 00 00 00 00
3. 00 00 00 00 00 01 06 00
4. <PacketHeader>
5. <Type>
6. <BYTE>0E </BYTE>
7. </Type>
8. <Status>
9. <BYTE>01 </BYTE>
10. </Status>
11. <Length>
12. <BYTE>00 </BYTE>
13. <BYTE>20 </BYTE>
14. </Length>
15. <SPID>
16. <BYTE>00 </BYTE>
17. <BYTE>00 </BYTE>
18. </SPID>
19. <Packet>
20. <BYTE>01 </BYTE>
21. </Packet>
22. <Window>
23. <BYTE>00 </BYTE>
24. </Window>
25. </PacketHeader>
26. <PacketData>
27. <TransMgrReq>
28. <All\_HEADERS>
29. <TotalLength>
30. <DWORD>16 00 00 00 </DWORD>
31. </TotalLength>
32. <Header>
33. <HeaderLength>
34. <DWORD>12 00 00 00 </DWORD>
35. </HeaderLength>
36. <HeaderType>
37. <USHORT>02 00 </USHORT>
38. </HeaderType>
39. <HeaderData>
40. <MARS>
41. <TransactionDescriptor>
42. <ULONGLONG>00 00 00 00 00 00 00 01 </ULONGLONG>
43. </TransactionDescriptor>
44. <OutstandingRequestCount>
45. <DWORD>00 00 00 00 </DWORD>
46. </OutstandingRequestCount>
47. </MARS>
48. </HeaderData>
49. </Header>
50. </All\_HEADERS>
51. <RequestType>
52. <USHORT>16 00 </USHORT>
53. </RequestType>
54. <RequestPayload>
55. <TM\_PROMOTE\_XACT>
56. </TM\_PROMOTE\_XACT>
57. </RequestPayload>
58. </TransMgrReq>
59. </PacketData>

## TVP Insert Statement

TVP insert statement sent from client to server:

1. 03 01 00 52 00 00 01 00 16 00 00 00
2. 12 00 00 00 02 00 00 00 00 00 00 00
3. 00 00 00 00 00 01 03 00 66 00 6F 00
4. 6F 00 00 00 00 00 F3 00 03 64 00 62
5. 00 6F 00 07 74 00 76 00 70 00 74 00
6. 79 00 70 00 65 00 01 00 00 00 00 00
7. 00 00 26 01 00 00 01 01 02 00
9. <tds version="katmai">
10. <PacketData>
11. <RPCRequest>
12. <RPCReqBatch>
13. <NameLenProcID>
14. <ProcName>
15. <US\_UNICODE>
16. <USHORTLEN>
17. <USHORT>03 00 </USHORT>
18. </USHORTLEN>
19. <BYTES ascii="f.o.o.">66 00 6F 00 6F 00 </BYTES>
20. </US\_UNICODE>
21. </ProcName>
22. </NameLenProcID>
23. <OptionFlags>
24. <fWithRecomp>
25. <BIT>false</BIT>
26. </fWithRecomp>
27. <fNoMetaData>
28. <BIT>false</BIT>
29. </fNoMetaData>
30. <fReuseMetaData>
31. <BIT>false</BIT>
32. </fResuseMetaData>
33. </OptionFlags>
34. <ParameterData>
35. <ParamMetaData>
36. <B\_UNICODE>
37. <BYTELEN>
38. <BYTE>00 </BYTE>
39. </BYTELEN>
40. <BYTES ascii="">
41. </BYTES>
42. </B\_UNICODE>
43. <StatusFlags>
44. <fByRefValue>
45. <BIT>false</BIT>
46. </fByRefValue>
47. <fDefaultValue>
48. <BIT>false</BIT>
49. </fDefaultValue>
50. <fCookie>
51. <BIT>false</BIT>
52. </fCookie>
53. </StatusFlags>
54. <TYPE\_INFO>
55. <TVP\_TYPE\_INFO>
56. <TVP\_TYPE>
57. <BYTE>F3 </BYTE>
58. </TVP\_TYPE>
59. <TVP\_TYPE\_NAME>
60. <DbName>
61. <B\_UNICODE></B\_UNICODE>
62. </DbName>
63. <OwningSchema>
64. <B\_UNICODE>dbo</B\_UNICODE>
65. </OwningSchema>
66. <TypeName>
67. <B\_UNICODE>tvptype</B\_UNICODE>
68. </TypeName>
69. </TVP\_TYPE\_NAME>
70. <TVP\_COLMETADATA>
71. <Count>
72. <USHORT>01 00 </USHORT>
73. </Count>
74. <TvpColumnMetaData>
75. <UserType>
76. <ULONG>00 00 00 00 </ULONG>
77. </UserType>
78. <Flags>
79. <USHORT>00 00 </USHORT>
80. </Flags>
81. <TYPE\_INFO>
82. <VARLENTYPE>
83. <BYTELEN\_TYPE>
84. <BYTE>26 </BYTE>
85. </BYTELEN\_TYPE>
86. </VARLENTYPE>
87. <TYPE\_VARLEN>
88. <BYTELEN>
89. <BYTE>01 </BYTE>
90. </BYTELEN>
91. </TYPE\_VARLEN>
92. </TYPE\_INFO>
93. <ColName>
94. <B\_UNICODE>
95. <BYTELEN>
96. <BYTE>00 </BYTE>
97. </BYTELEN>
98. <BYTES ascii="">
99. </BYTES>
100. </B\_UNICODE>
101. </ColName>
102. </TvpColumnMetaData>
103. </TVP\_COLMETADATA>
104. <TVP\_END\_TOKEN>
105. <TokenType>
106. <BYTE>00 </BYTE>
107. </TokenType>
108. </TVP\_END\_TOKEN>
109. <TVP\_ROW>
110. <TokenType>
111. <BYTE>01 </BYTE>
112. </TokenType>
113. <AllColumnData>
114. <TYPE\_VARBYTE>
115. <TYPE\_VARLEN>
116. <BYTELEN>
117. <BYTE>01</BYTE>
118. </BYTELEN>
119. <BYTES>02</BYTES>
120. </TYPE\_VARLEN>
121. </TYPE\_VARBYTE>
122. </AllColumnData>
123. </TVP\_ROW>
124. <TVP\_END\_TOKEN>
125. <TokenType>
126. <BYTE>00 </BYTE>
127. </TokenType>
128. </TVP\_END\_TOKEN>
129. </TVP\_TYPE\_INFO>
130. </TYPE\_INFO>
131. </ParamMetaData>
132. <ParamLenData>
133. </ParamLenData>
134. </ParameterData>
135. </RPCReqBatch>
136. </RPCRequest>
137. </PacketData>
138. </tds>

## SparseColumn Select Statement

SparseColumn select statement sent from client to server:

1. 04 01 01 B9 00 00 01 00 81 02 00 00 00 00 00 09 00
2. 26 04 02 69 00 64 00 00 00 00 00 0B 04 F1 00 11 73
3. 00 70 00 61 00 72 00 73 00 65 00 50 00 72 00 6F 00
4. 70 00 65 00 72 00 74 00 79 00 53 00 65 00 74 00 D1
5. 04 01 00 00 00 FE FF FF FF FF FF FF FF 7A 00 00 00
6. 3C 00 73 00 70 00 61 00 72 00 73 00 65 00 50 00 72
7. 00 6F 00 70 00 31 00 3E 00 31 00 30 00 30 00 30 00
8. 3C 00 2F 00 73 00 70 00 61 00 72 00 73 00 65 00 50
9. 00 72 00 6F 00 70 00 31 00 3E 00 3C 00 73 00 70 00
10. 61 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
11. 00 3E 00 66 00 6F 00 6F 00 3C 00 2F 00 73 00 70 00
12. 61 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
13. 00 3E 00 00 00 00 00 D1 04 02 00 00 00 FE FF FF FF
14. FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61 00 72
15. 00 73 00 65 00 50 00 72 00 6F 00 70 00 31 00 3E 00
16. 31 00 30 00 30 00 30 00 3C 00 2F 00 73 00 70 00 61
17. 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 31
18. 00 3E 00 00 00 00 00 D1 04 03 00 00 00 FE FF FF
19. FF FF FF FF FF 3E 00 00 00 3C 00 73 00 70 00 61
20. 00 72 00 73 00 65 00 50 00 72 00 6F 00 70 00 32
21. 00 3E 00 61 00 62 00 63 00 64 00 3C 00 2F 00 73
22. 00 70 00 61 00 72 00 73 00 65 00 50 00 72 00 6F
23. 00 70 00 32 00 3E 00 00 00 00 00 FD 10 00 C1 00
24. 0A 00 00 00 00 00 00 00
25. <tds version="katmai">
26. <PacketHeader>
27. <Type>
28. <BYTE>04 </BYTE>
29. </Type>
30. <Status>
31. <BYTE>01 </BYTE>
32. </Status>
33. <Length>
34. <BYTE>01 </BYTE>
35. <BYTE>B9 </BYTE>
36. </Length>
37. <SPID>
38. <BYTE>00 </BYTE>
39. <BYTE>00 </BYTE>
40. </SPID>
41. <Packet>
42. <BYTE>01 </BYTE>
43. </Packet>
44. <Window>
45. <BYTE>00 </BYTE>
46. </Window>
47. </PacketHeader>
48. <PacketData>
49. <TableResponse>
50. <COLMETADATA>
51. <TokenType>
52. <BYTE>81 </BYTE>
53. </TokenType>
54. <Count>
55. <USHORT>02 00 </USHORT>
56. </Count>
57. <ColumnData>
58. <UserType>
59. <ULONG>00 00 00 00 </ULONG>
60. </UserType>
61. <Flags>
62. <USHORT>09 00 </USHORT>
63. </Flags>
64. <TYPE\_INFO>
65. <VARLENTYPE>
66. <BYTELEN\_TYPE>
67. <BYTE>26 </BYTE>
68. </BYTELEN\_TYPE>
69. </VARLENTYPE>
70. <TYPE\_VARLEN>
71. <BYTELEN>
72. <BYTE>04 </BYTE>
73. </BYTELEN>
74. </TYPE\_VARLEN>
75. </TYPE\_INFO>
76. <ColName>
77. <B\_UNICODE>
78. <BYTELEN>
79. <BYTE>02 </BYTE>
80. </BYTELEN>
81. <BYTES ascii="i.d.">69 00 64 00 </BYTES>
82. </B\_UNICODE>
83. </ColName>
84. </ColumnData>
85. <ColumnData>
86. <UserType>
87. <ULONG>00 00 00 00 </ULONG>
88. </UserType>
89. <Flags fSparseColumn="true">
90. <USHORT>0B 04 </USHORT>
91. </Flags>
92. <TYPE\_INFO>
93. <VARLENTYPE>
94. <USHORTLEN\_TYPE>
95. <BYTE>F1 </BYTE>
96. </USHORTLEN\_TYPE>
97. </VARLENTYPE>
98. <XML\_INFO>
99. <SCHEMA\_PRESENT>
100. <BYTE>00 </BYTE>
101. </SCHEMA\_PRESENT>
102. </XML\_INFO>
103. </TYPE\_INFO>
104. <ColName>
105. <B\_UNICODE>
106. <BYTELEN>
107. <BYTE>11 </BYTE>
108. </BYTELEN>
109. <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>
110. </B\_UNICODE>
111. </ColName>
112. </ColumnData>
113. </COLMETADATA>
114. <ROW>
115. <TokenType>
116. <BYTE>D1 </BYTE>
117. </TokenType>
118. <TYPE\_VARBYTE>
119. <TYPE\_VARLEN>
120. <BYTELEN>
121. <BYTE>04 </BYTE>
122. </BYTELEN>
123. </TYPE\_VARLEN>
124. <BYTES>01 00 00 00 </BYTES>
125. </TYPE\_VARBYTE>
126. <TYPE\_VARBYTE>
127. <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>
128. </TYPE\_VARBYTE>
129. </ROW>
130. <ROW>
131. <TokenType>
132. <BYTE>D1 </BYTE>
133. </TokenType>
134. <TYPE\_VARBYTE>
135. <TYPE\_VARLEN>
136. <BYTELEN>
137. <BYTE>04 </BYTE>
138. </BYTELEN>
139. </TYPE\_VARLEN>
140. <BYTES>02 00 00 00 </BYTES>
141. </TYPE\_VARBYTE>
142. <TYPE\_VARBYTE>
143. <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>
144. </TYPE\_VARBYTE>
145. </ROW>
146. <ROW>
147. <TokenType>
148. <BYTE>D1 </BYTE>
149. </TokenType>
150. <TYPE\_VARBYTE>
151. <TYPE\_VARLEN>
152. <BYTELEN>
153. <BYTE>04 </BYTE>
154. </BYTELEN>
155. </TYPE\_VARLEN>
156. <BYTES>03 00 00 00 </BYTES>
157. </TYPE\_VARBYTE>
158. <TYPE\_VARBYTE>
159. <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>
160. </TYPE\_VARBYTE>
161. </ROW>
162. <NBCROW>
163. <TokenType>
164. <BYTE>D2 </BYTE>
165. </TokenType>
166. <NBCBitMap>
167. <BYTES>02 </BYTES>
168. </NBCBitMap>
169. <TYPE\_VARBYTE>
170. <TYPE\_VARLEN>
171. <BYTELEN>
172. <BYTE>04 </BYTE>
173. </BYTELEN>
174. </TYPE\_VARLEN>
175. <BYTES>04 00 00 00 </BYTES>
176. </TYPE\_VARBYTE>
177. </NBCROW>
178. <NBCROW>
179. <TokenType>
180. <BYTE>D2 </BYTE>
181. </TokenType>
182. <NBCBitMap>
183. <BYTES>02 </BYTES>
184. </NBCBitMap>
185. <TYPE\_VARBYTE>
186. <TYPE\_VARLEN>
187. <BYTELEN>
188. <BYTE>04 </BYTE>
189. </BYTELEN>
190. </TYPE\_VARLEN>
191. <BYTES>05 00 00 00 </BYTES>
192. </TYPE\_VARBYTE>
193. </NBCROW>
194. <NBCROW>
195. <TokenType>
196. <BYTE>D2 </BYTE>
197. </TokenType>
198. <NBCBitMap>
199. <BYTES>02 </BYTES>
200. </NBCBitMap>
201. <TYPE\_VARBYTE>
202. <TYPE\_VARLEN>
203. <BYTELEN>
204. <BYTE>04 </BYTE>
205. </BYTELEN>
206. </TYPE\_VARLEN>
207. <BYTES>06 00 00 00 </BYTES>
208. </TYPE\_VARBYTE>
209. </NBCROW>
210. <NBCROW>
211. <TokenType>
212. <BYTE>D2 </BYTE>
213. </TokenType>
214. <NBCBitMap>
215. <BYTES>02 </BYTES>
216. </NBCBitMap>
217. <TYPE\_VARBYTE>
218. <TYPE\_VARLEN>
219. <BYTELEN>
220. <BYTE>04 </BYTE>
221. </BYTELEN>
222. </TYPE\_VARLEN>
223. <BYTES>07 00 00 00 </BYTES>
224. </TYPE\_VARBYTE>
225. </NBCROW>
226. <NBCROW>
227. <TokenType>
228. <BYTE>D2 </BYTE>
229. </TokenType>
230. <NBCBitMap>
231. <BYTES>02 </BYTES>
232. </NBCBitMap>
233. <TYPE\_VARBYTE>
234. <TYPE\_VARLEN>
235. <BYTELEN>
236. <BYTE>04 </BYTE>
237. </BYTELEN>
238. </TYPE\_VARLEN>
239. <BYTES>08 00 00 00 </BYTES>
240. </TYPE\_VARBYTE>
241. </NBCROW>
242. <NBCROW>
243. <TokenType>
244. <BYTE>D2 </BYTE>
245. </TokenType>
246. <NBCBitMap>
247. <BYTES>02 </BYTES>
248. </NBCBitMap>
249. <TYPE\_VARBYTE>
250. <TYPE\_VARLEN>
251. <BYTELEN>
252. <BYTE>04 </BYTE>
253. </BYTELEN>
254. </TYPE\_VARLEN>
255. <BYTES>09 00 00 00 </BYTES>
256. </TYPE\_VARBYTE>
257. </NBCROW>
258. <NBCROW>
259. <TokenType>
260. <BYTE>D2 </BYTE>
261. </TokenType>
262. <NBCBitMap>
263. <BYTES>02 </BYTES>
264. </NBCBitMap>
265. <TYPE\_VARBYTE>
266. <TYPE\_VARLEN>
267. <BYTELEN>
268. <BYTE>04 </BYTE>
269. </BYTELEN>
270. </TYPE\_VARLEN>
271. <BYTES>0A 00 00 00 </BYTES>
272. </TYPE\_VARBYTE>
273. </NBCROW>
274. <DONE>
275. <TokenType>
276. <BYTE>FD </BYTE>
277. </TokenType>
278. <Status>
279. <USHORT>10 00 </USHORT>
280. </Status>
281. <CurCmd>
282. <USHORT>C1 00 </USHORT>
283. </CurCmd>
284. <DoneRowCount>
285. <LONGLONG>0A 00 00 00 00 00 00 00 </LONGLONG>
286. </DoneRowCount>
287. </DONE>
288. </TableResponse>
289. </PacketData>
290. </tds>

## FeatureExt with SESSIONRECOVERY Feature Data

A login message that contains FeatureExt data for the SESSIONRECOVERY feature:

1. 10 01 01 0D 00 00 01 00 05 01 00 00 04 00 00 74
2. 00 10 00 00 00 00 00 07 00 01 00 00 00 00 00 00
3. E0 03 00 10 E0 01 00 00 09 04 00 00 5E 00 00 00
4. 5E 00 02 00 62 00 08 00 72 00 07 00 80 00 00 00
5. 80 00 04 00 84 00 04 00 8C 00 00 00 8C 00 06 00
6. 00 50 8B E2 B7 8F 98 00 00 00 98 00 00 00 98 00
7. 00 00 00 00 00 00 73 00 61 00 00 00 00 00 00 00
8. 00 00 00 00 00 00 00 00 00 00 4F 00 53 00 51 00
9. 4C 00 2D 00 33 00 32 00 98 00 00 00 4F 00 44 00
10. 42 00 43 00 74 00 65 00 6D 00 70 00 64 00 62 00
11. 01 67 00 00 00 56 00 00 00 06 6D 00 61 00 73 00
12. 74 00 65 00 72 00 05 09 04 D0 00 34 0A 75 00 73
13. 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68
14. 00 00 09 00 60 81 14 FF E7 FF FF 00 02 02 07 01
15. 04 01 00 05 04 FF FF FF FF 06 01 00 07 01 02 08
16. 08 00 00 00 00 00 00 00 00 09 04 FF FF FF FF 09
17. 00 00 00 00 00 00 09 04 28 23 00 00 FF
18. <tds version="latest">
19. <PacketHeader>
20. <Type>
21. <BYTE>10 </BYTE>
22. </Type>
23. <Status>
24. <BYTE>01 </BYTE>
25. </Status>
26. <Length>
27. <BYTE>01 </BYTE>
28. <BYTE>0D </BYTE>
29. </Length>
30. <SPID>
31. <BYTE>00 </BYTE>
32. <BYTE>00 </BYTE>
33. </SPID>
34. <Packet>
35. <BYTE>01 </BYTE>
36. </Packet>
37. <Window>
38. <BYTE>00 </BYTE>
39. </Window>
40. </PacketHeader>
41. <PacketData>
42. <Login7>
43. <Length>
44. <DWORD>05 01 00 00 </DWORD>
45. </Length>
46. <TDSVersion>
47. <DWORD>04 00 00 74 </DWORD>
48. </TDSVersion>
49. <PacketSize>
50. <DWORD>00 10 00 00 </DWORD>
51. </PacketSize>
52. <ClientProgVer>
53. <DWORD>00 00 00 07 </DWORD>
54. </ClientProgVer>
55. <ClientPID>
56. <DWORD>00 01 00 00 </DWORD>
57. </ClientPID>
58. <ConnectionID>
59. <DWORD>00 00 00 00 </DWORD>
60. </ConnectionID>
61. <OptionFlags1>
62. <BYTE>E0 </BYTE>
63. </OptionFlags1>
64. <OptionFlags2>
65. <BYTE>03 </BYTE>
66. </OptionFlags2>
67. <TypeFlags>
68. <BYTE>00 </BYTE>
69. </TypeFlags>
70. <OptionFlags3>
71. <BYTE>10 </BYTE>
72. </OptionFlags3>
73. <ClientTimZone>
74. <DWORD>E0 01 00 00 </DWORD>
75. </ClientTimZone>
76. <ClientLCID>
77. <DWORD>09 04 00 00 </DWORD>
78. </ClientLCID>
79. <OffsetLength>
80. <ibHostName>
81. <USHORT>5E 00 </USHORT>
82. </ibHostName>
83. <cchHostName>
84. <USHORT>00 00 </USHORT>
85. </cchHostName>
86. <ibUserName>
87. <USHORT>5E 00 </USHORT>
88. </ibUserName>
89. <cchUserName>
90. <USHORT>02 00 </USHORT>
91. </cchUserName>
92. <ibPassword>
93. <USHORT>62 00 </USHORT>
94. </ibPassword>
95. <cchPassword>
96. <USHORT>08 00 </USHORT>
97. </cchPassword>
98. <ibAppName>
99. <USHORT>72 00 </USHORT>
100. </ibAppName>
101. <cchAppName>
102. <USHORT>07 00 </USHORT>
103. </cchAppName>
104. <ibServerName>
105. <USHORT>80 00 </USHORT>
106. </ibServerName>
107. <cchServerName>
108. <USHORT>00 00 </USHORT>
109. </cchServerName>
110. <ibExtension>
111. <USHORT>80 00 </USHORT>
112. </ibExtension>
113. <cbExtension>
114. <USHORT>04 00 </USHORT>
115. </cbExtension>
116. <ibCltIntName>
117. <USHORT>84 00 </USHORT>
118. </ibCltIntName>
119. <cchCltIntName>
120. <USHORT>04 00 </USHORT>
121. </cchCltIntName>
122. <ibLanguage>
123. <USHORT>8C 00 </USHORT>
124. </ibLanguage>
125. <cchLanguage>
126. <USHORT>00 00 </USHORT>
127. </cchLanguage>
128. <ibDatabase>
129. <USHORT>8C 00 </USHORT>
130. </ibDatabase>
131. <cchDatabase>
132. <USHORT>06 00 </USHORT>
133. </cchDatabase>
134. <ClientID>
135. <BYTES>00 50 8B E2 B7 8F </BYTES>
136. </ClientID>
137. <ibSSPI>
138. <USHORT>98 00 </USHORT>
139. </ibSSPI>
140. <cbSSPI>
141. <USHORT>00 00 </USHORT>
142. </cbSSPI>
143. <ibAtchDBFile>
144. <USHORT>98 00 </USHORT>
145. </ibAtchDBFile>
146. <cchAtchDBFile>
147. <USHORT>00 00 </USHORT>
148. </cchAtchDBFile>
149. <ibChangePassword>
150. <USHORT>98 00 </USHORT>
151. </ibChangePassword>
152. <cchChangePassword>
153. <USHORT>00 00 </USHORT>
154. </cchChangePassword>
155. <cbSSPILong>
156. <LONG>00 00 00 00 </LONG>
157. </cbSSPILong>
158. </OffsetLength>
159. <Data>
160. <BYTES>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 </BYTES>
161. </Data>
162. <FeatureExt>
163. <FeatureOpt>
164. <FeatureId>
165. <BYTE>01 </BYTE>
166. </FeatureId>
167. <FeatureDataLen>
168. <DWORD>67 00 00 00 </DWORD>
169. </FeatureDataLen>
170. <FeatureData>
171. <InitSessionRecoveryData>
172. <Length>
173. <DWORD>56 00 00 00 </DWORD>
174. </Length>
175. <RecoveryDatabase>
176. <B\_VARCHAR>
177. <BYTE>06 </BYTE>
178. <BYTES ascii="m.a.s.t.e.r.">6D 00 61 00 73 00 74 00 65 00 72 00 </BYTES>
179. </B\_VARCHAR>
180. </RecoveryDatabase>
181. <RecoveryCollation>
182. <BYTELEN>
183. <BYTE>05 </BYTE>
184. </BYTELEN>
185. <BYTES>09 04 D0 00 34 </BYTES>
186. </RecoveryCollation>
187. <RecoveryLanguage>
188. <B\_VARCHAR>
189. <BYTE>0A </BYTE>
190. <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>
191. </B\_VARCHAR>
192. </RecoveryLanguage>
193. <SessionStateDataSet>
194. <SessionStateData>
195. <StateId>
196. <BYTE>00 </BYTE>
197. </StateId>
198. <StateLen>
199. <BYTE>09 </BYTE>
200. </StateLen>
201. <StateValue>
202. <BYTES>00 60 81 14 FF E7 FF FF 00 </BYTES>
203. </StateValue>
204. </SessionStateData>
205. <SessionStateData>
206. <StateId>
207. <BYTE>02 </BYTE>
208. </StateId>
209. <StateLen>
210. <BYTE>02 </BYTE>
211. </StateLen>
212. <StateValue>
213. <BYTES>07 01 </BYTES>
214. </StateValue>
215. </SessionStateData>
216. <SessionStateData>
217. <StateId>
218. <BYTE>04 </BYTE>
219. </StateId>
220. <StateLen>
221. <BYTE>01 </BYTE>
222. </StateLen>
223. <StateValue>
224. <BYTES>00 </BYTES>
225. </StateValue>
226. </SessionStateData>
227. <SessionStateData>
228. <StateId>
229. <BYTE>05 </BYTE>
230. </StateId>
231. <StateLen>
232. <BYTE>04 </BYTE>
233. </StateLen>
234. <StateValue>
235. <BYTES>FF FF FF FF </BYTES>
236. </StateValue>
237. </SessionStateData>
238. <SessionStateData>
239. <StateId>
240. <BYTE>06 </BYTE>
241. </StateId>
242. <StateLen>
243. <BYTE>01 </BYTE>
244. </StateLen>
245. <StateValue>
246. <BYTES>00 </BYTES>
247. </StateValue>
248. </SessionStateData>
249. <SessionStateData>
250. <StateId>
251. <BYTE>07 </BYTE>
252. </StateId>
253. <StateLen>
254. <BYTE>01 </BYTE>
255. </StateLen>
256. <StateValue>
257. <BYTES>02 </BYTES>
258. </StateValue>
259. </SessionStateData>
260. <SessionStateData>
261. <StateId>
262. <BYTE>08 </BYTE>
263. </StateId>
264. <StateLen>
265. <BYTE>08 </BYTE>
266. </StateLen>
267. <StateValue>
268. <BYTES>00 00 00 00 00 00 00 00 </BYTES>
269. </StateValue>
270. </SessionStateData>
271. <SessionStateData>
272. <StateId>
273. <BYTE>09 </BYTE>
274. </StateId>
275. <StateLen>
276. <BYTE>04 </BYTE>
277. </StateLen>
278. <StateValue>
279. <BYTES>FF FF FF FF </BYTES>
280. </StateValue>
281. </SessionStateData>
282. </SessionStateDataSet>
283. </InitSessionRecoveryData>
284. <SessionRecoveryDataToBe>
285. <Length>
286. <DWORD>09 00 00 00 </DWORD>
287. </Length>
288. <RecoveryDatabase>
289. <B\_VARCHAR>
290. <BYTE>00 </BYTE>
291. <BYTES ascii="">
292. </BYTES>
293. </B\_VARCHAR>
294. </RecoveryDatabase>
295. <RecoveryCollation>
296. <BYTELEN>
297. <BYTE>00 </BYTE>
298. </BYTELEN>
299. <BYTES>
300. </BYTES>
301. </RecoveryCollation>
302. <RecoveryLanguage>
303. <B\_VARCHAR>
304. <BYTE>00 </BYTE>
305. <BYTES ascii="">
306. </BYTES>
307. </B\_VARCHAR>
308. </RecoveryLanguage>
309. <SessionStateDataSet>
310. <SessionStateData>
311. <StateId>
312. <BYTE>09 </BYTE>
313. </StateId>
314. <StateLen>
315. <BYTE>04 </BYTE>
316. </StateLen>
317. <StateValue>
318. <BYTES>28 23 00 00 </BYTES>
319. </StateValue>
320. </SessionStateData>
321. </SessionStateDataSet>
322. </SessionRecoveryDataToBe>
323. </FeatureData>
324. </FeatureOpt>
325. <FeatureOpt>
326. <TERMINATOR>
327. <BYTE>FF </BYTE>
328. </TERMINATOR>
329. </FeatureOpt>
330. </FeatureExt>
331. </Login7>
332. </PacketData>
333. </tds>

## FeatureExtAck with SESSIONRECOVERY Feature Data

A login response message that contains FeatureExtAck data for the SESSIONRECOVERY feature:

1. 04 01 01 96 00 34 01 00 E3 1B 00 01 06 6D 00 61
2. 00 73 00 74 00 65 00 72 00 06 6D 00 61 00 73 00
3. 74 00 65 00 72 00 AB 58 00 45 16 00 00 02 00 25
4. 00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
5. 00 64 00 61 00 74 00 61 00 62 00 61 00 73 00 65
6. 00 20 00 63 00 6F 00 6E 00 74 00 65 00 78 00 74
7. 00 20 00 74 00 6F 00 20 00 27 00 6D 00 61 00 73
8. 00 74 00 65 00 72 00 27 00 2E 00 00 00 00 00 00
9. 00 E3 08 00 07 05 09 04 D0 00 34 00 E3 17 00 02
10. 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69
11. 00 73 00 68 00 00 AB 5C 00 47 16 00 00 01 00 27
12. 00 43 00 68 00 61 00 6E 00 67 00 65 00 64 00 20
13. 00 6C 00 61 00 6E 00 67 00 75 00 61 00 67 00 65
14. 00 20 00 73 00 65 00 74 00 74 00 69 00 6E 00 67
15. 00 20 00 74 00 6F 00 20 00 75 00 73 00 5F 00 65
16. 00 6E 00 67 00 6C 00 69 00 73 00 68 00 2E 00 00
17. 00 00 00 00 00 AD 36 00 01 74 00 00 04 16 4D 00
18. 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00
19. 20 00 53 00 51 00 4C 00 20 00 53 00 65 00 72 00
20. 76 00 65 00 72 00 00 00 00 00 0B 00 08 C3 E3 13
21. 00 04 04 34 00 30 00 39 00 36 00 04 34 00 30 00
22. 39 00 36 00 AE 01 2E 00 00 00 00 09 00 60 81 14
23. FF E7 FF FF 00 02 02 07 01 04 01 00 05 04 FF FF
24. FF FF 06 01 00 07 01 02 08 08 00 00 00 00 00 00
25. 00 00 09 04 28 23 00 00 FF FD 00 00 00 00 00 00
26. 00 00 00 00 00 00
27. <tds version="latest">
28. <PacketHeader>
29. <Type>
30. <BYTE>04 </BYTE>
31. </Type>
32. <Status>
33. <BYTE>01 </BYTE>
34. </Status>
35. <Length>
36. <BYTE>01 </BYTE>
37. <BYTE>96 </BYTE>
38. </Length>
39. <SPID>
40. <BYTE>00 </BYTE>
41. <BYTE>00 </BYTE>
42. </SPID>
43. <Packet>
44. <BYTE>01 </BYTE>
45. </Packet>
46. <Window>
47. <BYTE>00 </BYTE>
48. </Window>
49. </PacketHeader>
50. <PacketData>
51. <TableResponse>
52. <ENVCHANGE>
53. <TokenType>
54. <BYTE>E3 </BYTE>
55. </TokenType>
56. <Length>
57. <USHORT>1B 00 </USHORT>
58. </Length>
59. <EnvChangeData>
60. <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>
61. </EnvChangeData>
62. </ENVCHANGE>
63. <INFO>
64. <TokenType>
65. <BYTE>AB </BYTE>
66. </TokenType>
67. <Length>
68. <USHORT>58 00 </USHORT>
69. </Length>
70. <Number>
71. <LONG>45 16 00 00 </LONG>
72. </Number>
73. <State>
74. <BYTE>02 </BYTE>
75. </State>
76. <Class>
77. <BYTE>00 </BYTE>
78. </Class>
79. <MsgText>
80. <US\_UNICODE>
81. <USHORTLEN>
82. <USHORT>25 00 </USHORT>
83. </USHORTLEN>
84. <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>
85. </US\_UNICODE>
86. </MsgText>
87. <ServerName>
88. <B\_UNICODE>
89. <BYTELEN>
90. <BYTE>00 </BYTE>
91. </BYTELEN>
92. <BYTES ascii="">
93. </BYTES>
94. </B\_UNICODE>
95. </ServerName>
96. <ProcName>
97. <B\_UNICODE>
98. <BYTELEN>
99. <BYTE>00 </BYTE>
100. </BYTELEN>
101. <BYTES ascii="">
102. </BYTES>
103. </B\_UNICODE>
104. </ProcName>
105. <LineNumber>
106. <LONG>00 00 00 00 </LONG>
107. </LineNumber>
108. </INFO>
109. <ENVCHANGE>
110. <TokenType>
111. <BYTE>E3 </BYTE>
112. </TokenType>
113. <Length>
114. <USHORT>08 00 </USHORT>
115. </Length>
116. <EnvChangeData>
117. <BYTES>07 05 09 04 D0 00 34 00 </BYTES>
118. </EnvChangeData>
119. </ENVCHANGE>
120. <ENVCHANGE>
121. <TokenType>
122. <BYTE>E3 </BYTE>
123. </TokenType>
124. <Length>
125. <USHORT>17 00 </USHORT>
126. </Length>
127. <EnvChangeData>
128. <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>
129. </EnvChangeData>
130. </ENVCHANGE>
131. <INFO>
132. <TokenType>
133. <BYTE>AB </BYTE>
134. </TokenType>
135. <Length>
136. <USHORT>5C 00 </USHORT>
137. </Length>
138. <Number>
139. <LONG>47 16 00 00 </LONG>
140. </Number>
141. <State>
142. <BYTE>01 </BYTE>
143. </State>
144. <Class>
145. <BYTE>00 </BYTE>
146. </Class>
147. <MsgText>
148. <US\_UNICODE>
149. <USHORTLEN>
150. <USHORT>27 00 </USHORT>
151. </USHORTLEN>
152. <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>
153. </US\_UNICODE>
154. </MsgText>
155. <ServerName>
156. <B\_UNICODE>
157. <BYTELEN>
158. <BYTE>00 </BYTE>
159. </BYTELEN>
160. <BYTES ascii="">
161. </BYTES>
162. </B\_UNICODE>
163. </ServerName>
164. <ProcName>
165. <B\_UNICODE>
166. <BYTELEN>
167. <BYTE>00 </BYTE>
168. </BYTELEN>
169. <BYTES ascii="">
170. </BYTES>
171. </B\_UNICODE>
172. </ProcName>
173. <LineNumber>
174. <LONG>00 00 00 00 </LONG>
175. </LineNumber>
176. </INFO>
177. <LOGINACK>
178. <TokenType>
179. <BYTE>AD </BYTE>
180. </TokenType>
181. <Length>
182. <USHORT>36 00 </USHORT>
183. </Length>
184. <Interface>
185. <BYTE>01 </BYTE>
186. </Interface>
187. <TDSVersion>
188. <DWORD>FILTERED LATEST VERSION</DWORD>
189. </TDSVersion>
190. <ProgName>
191. <B\_UNICODE>
192. <BYTELEN>
193. <BYTE>16 </BYTE>
194. </BYTELEN>
195. <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>
196. </B\_UNICODE>
197. </ProgName>
198. <ProgVersion>
199. <DWORD>00 00 00 00 </DWORD>
200. </ProgVersion>
201. </LOGINACK>
202. <ENVCHANGE>
203. <TokenType>
204. <BYTE>E3 </BYTE>
205. </TokenType>
206. <Length>
207. <USHORT>13 00 </USHORT>
208. </Length>
209. <EnvChangeData>
210. <BYTES>04 04 34 00 30 00 39 00 36 00 04 34 00 30 00 39 00 36 00 </BYTES>
211. </EnvChangeData>
212. </ENVCHANGE>
213. <FEATUREEXTACK>
214. <TokenType>
215. <BYTE>AE </BYTE>
216. </TokenType>
217. <FeatureAckOpt>
218. <FeatureId>
219. <BYTE>01 </BYTE>
220. </FeatureId>
221. <FeatureAckDataLen>
222. <DWORD>2E 00 00 00 </DWORD>
223. </FeatureAckDataLen>
224. <SessionStateDataSet>
225. <SessionStateData>
226. <StateId>
227. <BYTE>00 </BYTE>
228. </StateId>
229. <StateLen>
230. <BYTE>09 </BYTE>
231. </StateLen>
232. <StateValue>
233. <BYTES>00 60 81 14 FF E7 FF FF 00 </BYTES>
234. </StateValue>
235. </SessionStateData>
236. <SessionStateData>
237. <StateId>
238. <BYTE>02 </BYTE>
239. </StateId>
240. <StateLen>
241. <BYTE>02 </BYTE>
242. </StateLen>
243. <StateValue>
244. <BYTES>07 01 </BYTES>
245. </StateValue>
246. </SessionStateData>
247. <SessionStateData>
248. <StateId>
249. <BYTE>04 </BYTE>
250. </StateId>
251. <StateLen>
252. <BYTE>01 </BYTE>
253. </StateLen>
254. <StateValue>
255. <BYTES>00 </BYTES>
256. </StateValue>
257. </SessionStateData>
258. <SessionStateData>
259. <StateId>
260. <BYTE>05 </BYTE>
261. </StateId>
262. <StateLen>
263. <BYTE>04 </BYTE>
264. </StateLen>
265. <StateValue>
266. <BYTES>FF FF FF FF </BYTES>
267. </StateValue>
268. </SessionStateData>
269. <SessionStateData>
270. <StateId>
271. <BYTE>06 </BYTE>
272. </StateId>
273. <StateLen>
274. <BYTE>01 </BYTE>
275. </StateLen>
276. <StateValue>
277. <BYTES>00 </BYTES>
278. </StateValue>
279. </SessionStateData>
280. <SessionStateData>
281. <StateId>
282. <BYTE>07 </BYTE>
283. </StateId>
284. <StateLen>
285. <BYTE>01 </BYTE>
286. </StateLen>
287. <StateValue>
288. <BYTES>02 </BYTES>
289. </StateValue>
290. </SessionStateData>
291. <SessionStateData>
292. <StateId>
293. <BYTE>08 </BYTE>
294. </StateId>
295. <StateLen>
296. <BYTE>08 </BYTE>
297. </StateLen>
298. <StateValue>
299. <BYTES>00 00 00 00 00 00 00 00 </BYTES>
300. </StateValue>
301. </SessionStateData>
302. <SessionStateData>
303. <StateId>
304. <BYTE>09 </BYTE>
305. </StateId>
306. <StateLen>
307. <BYTE>04 </BYTE>
308. </StateLen>
309. <StateValue>
310. <BYTES>28 23 00 00 </BYTES>
311. </StateValue>
312. </SessionStateData>
313. </SessionStateDataSet>
314. </FeatureAckOpt>
315. <FeatureAckOpt>
316. <TERMINATOR>
317. <BYTE>FF </BYTE>
318. </TERMINATOR>
319. </FeatureAckOpt>
320. </FEATUREEXTACK>
321. <DONE>
322. <TokenType>
323. <BYTE>FD </BYTE>
324. </TokenType>
325. <Status>
326. <USHORT>00 00 </USHORT>
327. </Status>
328. <CurCmd>
329. <USHORT>00 00 </USHORT>
330. </CurCmd>
331. <DoneRowCount>
332. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
333. </DoneRowCount>
334. </DONE>
335. </TableResponse>
336. </PacketData>
337. </tds>

## Table Response with SESSIONSTATE Token Data

A response message that contains SESSIONSTATE token data:

1. 04 01 00 32 00 34 01 00 FD 01 00 BE 00 00 00 00
2. 00 00 00 00 00 E4 0B 00 00 00 01 00 00 00 01 09
3. 04 FF FF FF FF FD 00 00 FD 00 00 00 00 00 00 00
4. 00 00
5. <tds version="latest">
6. <PacketHeader>
7. <Type>
8. <BYTE>04 </BYTE>
9. </Type>
10. <Status>
11. <BYTE>01 </BYTE>
12. </Status>
13. <Length>
14. <BYTE>00 </BYTE>
15. <BYTE>32 </BYTE>
16. </Length>
17. <SPID>
18. <BYTE>00 </BYTE>
19. <BYTE>00 </BYTE>
20. </SPID>
21. <Packet>
22. <BYTE>01 </BYTE>
23. </Packet>
24. <Window>
25. <BYTE>00 </BYTE>
26. </Window>
27. </PacketHeader>
28. <PacketData>
29. <TableResponse>
30. <DONE>
31. <TokenType>
32. <BYTE>FD </BYTE>
33. </TokenType>
34. <Status>
35. <USHORT>01 00 </USHORT>
36. </Status>
37. <CurCmd>
38. <USHORT>BE 00 </USHORT>
39. </CurCmd>
40. <DoneRowCount>
41. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
42. </DoneRowCount>
43. </DONE>
44. <SESSIONSTATE>
45. <TokenType>
46. <BYTE>E4 </BYTE>
47. </TokenType>
48. <Length>
49. <DWORD>0B 00 00 00 </DWORD>
50. </Length>
51. <SeqNo>
52. <DWORD>01 00 00 00 </DWORD>
53. </SeqNo>
54. <Status>
55. <BYTE>01 </BYTE>
56. </Status>
57. <SessionStateDataSet>
58. <SessionStateData>
59. <StateId>
60. <BYTE>09 </BYTE>
61. </StateId>
62. <StateLen>
63. <BYTE>04 </BYTE>
64. </StateLen>
65. <StateValue>
66. <BYTES>FF FF FF FF </BYTES>
67. </StateValue>
68. </SessionStateData>
69. </SessionStateDataSet>
70. </SESSIONSTATE>
71. <DONE>
72. <TokenType>
73. <BYTE>FD </BYTE>
74. </TokenType>
75. <Status>
76. <USHORT>00 00 </USHORT>
77. </Status>
78. <CurCmd>
79. <USHORT>FD 00 </USHORT>
80. </CurCmd>
81. <DoneRowCount>
82. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
83. </DoneRowCount>
84. </DONE>
85. </TableResponse>
86. </PacketData>
87. </tds>

## 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](#Section_e48ebc472e954970a116da45862af90b).

### Sending a SQL Batch

In this example, a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) is sent to the server and the results are sent to the client. The SQL statement is as follows:

1. SQLStatement = select name, empid from employees
2. update employees set salary = salary \* 1.1
3. select name from employees where department = 'HR'
4. Client: SQLStatement
5. Server: COLMETADATA data stream
6. ROW data stream
7. .
8. .
9. ROW data stream
10. DONE data stream (with DONE\_COUNT & DONE\_MORE
11. bits set)
12. DONE data stream (for UPDATE, with DONE\_COUNT &
13. DONE\_MORE bits set)
14. COLMETADATA data stream
15. ROW data stream
16. .
17. .
18. ROW data stream
19. DONE data stream (with DONE\_COUNT bit set)

### Out-of-Band Attention Signal

In this example, a [**SQL statement**](#gt_dc5ca224-43ec-4b44-9dba-726d6fd6057d) 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.

1. Client: select name, empid from employees
2. Server: COLMETADATA data stream
3. ROW data stream
4. .
5. .
6. ROW data stream
7. 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.]

1. Server: DONE data stream (with DONE\_ATTN bit set)

## FeatureExt with AZURESQLSUPPORT Feature Data

A login message that contains FeatureExt data for the AZURESQLSUPPORT feature:

1. 10 01 01 C7 00 00 01 00 BF 01 00 00 04 00 00 74
2. 40 1F 00 00 00 00 00 06 98 1D 00 00 00 00 00 00
3. E0 03 20 10 00 00 00 00 00 00 00 00 5E 00 0C 00
4. 76 00 07 00 84 00 08 00 94 00 1C 00 CC 00 4A 00
5. 60 01 04 00 64 01 1C 00 9C 01 00 00 9C 01 06 00
6. C2 CC 3D 20 B7 AB A8 01 00 00 A8 01 00 00 A8 01
7. 00 00 00 00 00 00 5A 00 4C 00 49 00 4E 00 36 00
8. 43 00 4C 00 49 00 45 00 4E 00 54 00 32 00 63 00
9. 6C 00 6F 00 75 00 64 00 73 00 61 00 00 00 00 00
10. 00 00 00 00 00 00 00 00 00 00 00 00 2E 00 4E 00
11. 65 00 74 00 20 00 53 00 71 00 6C 00 43 00 6C 00
12. 69 00 65 00 6E 00 74 00 20 00 44 00 61 00 74 00
13. 61 00 20 00 50 00 72 00 6F 00 76 00 69 00 64 00
14. 65 00 72 00 65 00 32 00 66 00 38 00 38 00 37 00
15. 36 00 61 00 64 00 36 00 35 00 38 00 2E 00 6C 00
16. 6F 00 63 00 61 00 6C 00 2E 00 6F 00 6E 00 65 00
17. 62 00 6F 00 78 00 2E 00 63 00 6F 00 6E 00 74 00
18. 72 00 6F 00 6C 00 2E 00 7A 00 6C 00 69 00 6E 00
19. 68 00 65 00 6B 00 61 00 36 00 64 00 65 00 76 00
20. 34 00 2E 00 6F 00 6E 00 65 00 62 00 6F 00 78 00
21. 2E 00 78 00 64 00 62 00 2E 00 6D 00 73 00 63 00
22. 64 00 73 00 2E 00 63 00 6F 00 6D 00 2C 00 33 00
23. 37 00 30 00 30 00 38 00 A8 01 00 00 2E 00 4E 00
24. 65 00 74 00 20 00 53 00 71 00 6C 00 43 00 6C 00
25. 69 00 65 00 6E 00 74 00 20 00 44 00 61 00 74 00
26. 61 00 20 00 50 00 72 00 6F 00 76 00 69 00 64 00
27. 65 00 72 00 74 00 65 00 73 00 74 00 64 00 62 00
28. 01 00 00 00 00 04 01 00 00 00 01 05 00 00 00 00
29. 08 01 00 00 00 01 FF
30. <tds version="latest">
31. <PacketHeader>
32. <Type>
33. <BYTE>10 </BYTE>
34. </Type>
35. <Status>
36. <BYTE>01 </BYTE>
37. </Status>
38. <Length>
39. <BYTE>01 </BYTE>
40. <BYTE>C7 </BYTE>
41. </Length>
42. <SPID>
43. <BYTE>00 </BYTE>
44. <BYTE>00 </BYTE>
45. </SPID>
46. <Packet>
47. <BYTE>01 </BYTE>
48. </Packet>
49. <Window>
50. <BYTE>00 </BYTE>
51. </Window>
52. </PacketHeader>
53. <PacketData>
54. <Login7>
55. <Length>
56. <DWORD>BF 01 00 00 </DWORD>
57. </Length>
58. <TDSVersion>
59. <DWORD>04 00 00 74 </DWORD>
60. </TDSVersion>
61. <PacketSize>
62. <DWORD>40 1F 00 00 </DWORD>
63. </PacketSize>
64. <ClientProgVer>
65. <DWORD>00 00 00 06 </DWORD>
66. </ClientProgVer>
67. <ClientPID>
68. <DWORD>96 1D 00 00 </DWORD>
69. </ClientPID>
70. <ConnectionID>
71. <DWORD>00 00 00 00 </DWORD>
72. </ConnectionID>
73. <OptionFlags1>
74. <BYTE>E0 </BYTE>
75. </OptionFlags1>
76. <OptionFlags2>
77. <BYTE>03 </BYTE>
78. </OptionFlags2>
79. <TypeFlags>
80. <BYTE>20 </BYTE>
81. </TypeFlags>
82. <OptionFlags3>
83. <BYTE>10 </BYTE>
84. </OptionFlags3>
85. <ClientTimZone>
86. <DWORD>00 00 00 00 </DWORD>
87. </ClientTimZone>
88. <ClientLCID>
89. <DWORD>00 00 00 00 </DWORD>
90. </ClientLCID>
91. <OffsetLength>
92. <ibHostName>
93. <USHORT>5E 00 </USHORT>
94. </ibHostName>
95. <cchHostName>
96. <USHORT>0C 00 </USHORT>
97. </cchHostName>
98. <ibUserName>
99. <USHORT>76 00 </USHORT>
100. </ibUserName>
101. <cchUserName>
102. <USHORT>07 00 </USHORT>
103. </cchUserName>
104. <ibPassword>
105. <USHORT>84 00 </USHORT>
106. </ibPassword>
107. <cchPassword>
108. <USHORT>08 00 </USHORT>
109. </cchPassword>
110. <ibAppName>
111. <USHORT>94 00 </USHORT>
112. </ibAppName>
113. <cchAppName>
114. <USHORT>1C 00 </USHORT>
115. </cchAppName>
116. <ibServerName>
117. <USHORT>CC 00 </USHORT>
118. </ibServerName>
119. <cchServerName>
120. <USHORT>4A 00 </USHORT>
121. </cchServerName>
122. <ibExtension>
123. <USHORT>60 01 </USHORT>
124. </ibExtension>
125. <cbExtension>
126. <USHORT>04 00 </USHORT>
127. </cbExtension>
128. <ibCltIntName>
129. <USHORT>64 01 </USHORT>
130. </ibCltIntName>
131. <cchCltIntName>
132. <USHORT>1C 00 </USHORT>
133. </cchCltIntName>
134. <ibLanguage>
135. <USHORT>9C 01 </USHORT>
136. </ibLanguage>
137. <cchLanguage>
138. <USHORT>00 00 </USHORT>
139. </cchLanguage>
140. <ibDatabase>
141. <USHORT>9C 01 </USHORT>
142. </ibDatabase>
143. <cchDatabase>
144. <USHORT>06 00 </USHORT>
145. </cchDatabase>
146. <ClientID>
147. <BYTES>C2 CC 3D 20 B7 AB </BYTES>
148. </ClientID>
149. <ibSSPI>
150. <USHORT>AB 01 </USHORT>
151. </ibSSPI>
152. <cbSSPI>
153. <USHORT>00 00 </USHORT>
154. </cbSSPI>
155. <ibAtchDBFile>
156. <USHORT>AB 01 </USHORT>
157. </ibAtchDBFile>
158. <cchAtchDBFile>
159. <USHORT>00 00 </USHORT>
160. </cchAtchDBFile>
161. <ibChangePassword>
162. <USHORT>AB 01 </USHORT>
163. </ibChangePassword>
164. <cchChangePassword>
165. <USHORT>00 00 </USHORT>
166. </cchChangePassword>
167. <cbSSPILong>
168. <LONG>00 00 00 00 </LONG>
169. </cbSSPILong>
170. </OffsetLength>
171. <Data>
172. <BYTES>
173. 5A 00 4C 00 49 00 4E 00 36 00
174. 43 00 4C 00 49 00 45 00 4E 00 54 00 32 00 63 00
175. 6C 00 6F 00 75 00 64 00 73 00 61 00 00 00 00 00
176. 00 00 00 00 00 00 00 00 00 00 00 00 2E 00 4E 00
177. 65 00 74 00 20 00 53 00 71 00 6C 00 43 00 6C 00
178. 69 00 65 00 6E 00 74 00 20 00 44 00 61 00 74 00
179. 61 00 20 00 50 00 72 00 6F 00 76 00 69 00 64 00
180. 65 00 72 00 65 00 32 00 66 00 38 00 38 00 37 00
181. 36 00 61 00 64 00 36 00 35 00 38 00 2E 00 6C 00
182. 6F 00 63 00 61 00 6C 00 2E 00 6F 00 6E 00 65 00
183. 62 00 6F 00 78 00 2E 00 63 00 6F 00 6E 00 74 00
184. 72 00 6F 00 6C 00 2E 00 7A 00 6C 00 69 00 6E 00
185. 68 00 65 00 6B 00 61 00 36 00 64 00 65 00 76 00
186. 34 00 2E 00 6F 00 6E 00 65 00 62 00 6F 00 78 00
187. 2E 00 78 00 64 00 62 00 2E 00 6D 00 73 00 63 00
188. 64 00 73 00 2E 00 63 00 6F 00 6D 00 2C 00 33 00
189. 37 00 30 00 30 00 38 00 A8 01 00 00 2E 00 4E 00
190. 65 00 74 00 20 00 53 00 71 00 6C 00 43 00 6C 00
191. 69 00 65 00 6E 00 74 00 20 00 44 00 61 00 74 00
192. 61 00 20 00 50 00 72 00 6F 00 76 00 69 00 64 00
193. 65 00 72 00 74 00 65 00 73 00 74 00 64 00 62 00
194. </BYTES>
195. </Data>
196. <FeatureExt>
197. <FeatureOpt>
198. <FeatureId>
199. <BYTE>01 </BYTE>
200. </FeatureId>
201. <FeatureDataLen>
202. <DWORD>00 00 00 00 </DWORD>
203. </FeatureDataLen>
204. <FeatureId>
205. <BYTE>04 </BYTE>
206. </FeatureId>
207. <FeatureDataLen>
208. <DWORD>01 00 00 00 </DWORD>
209. </FeatureDataLen>
210. <FeatureData>
211. <BYTE>01</BYTE>
212. </FeatureData>
213. <FeatureId>
214. <BYTE>05 </BYTE>
215. </FeatureId>
216. <FeatureDataLen>
217. <DWORD>00 00 00 00 </DWORD>
218. </FeatureDataLen>
219. <FeatureId>
220. <AZURESQLSUPPORT>
221. <BYTE>08 </BYTE>
222. </AZURESQLSUPPORT>
223. </FeatureId>
224. <FeatureDataLen>
225. <DWORD>01 00 00 00 </DWORD>
226. </FeatureDataLen>
227. <FeatureData>
228. <BYTE>01</BYTE>
229. </FeatureData>
230. <TERMINATOR>
231. <BYTE>FF </BYTE>
232. </TERMINATOR>
233. </FeatureOpt>
234. </FeatureExt>
235. </Login7>
236. </PacketData>
237. </tds>

## FeatureExtAck with AZURESQLSUPPORT Feature Data

A login response message that contains FeatureExtAck data for the AZURESQLSUPPORT feature:

1. 04 01 02 C3 00 77 01 00 FF 11 00 C1 00 01 00 00
2. 00 00 00 00 00 FF 11 00 C1 00 00 00 00 00 00 00
3. 00 00 FF 01 00 C0 00 00 00 00 00 00 00 00 00 FF
4. 11 00 C1 00 01 00 00 00 00 00 00 00 FF 11 00 C1
5. 00 00 00 00 00 00 00 00 00 FF 01 00 C0 00 00 00
6. 00 00 00 00 00 00 FF 11 00 C1 00 01 00 00 00 00
7. 00 00 00 FF 11 00 C1 00 01 00 00 00 00 00 00 00
8. FF 11 00 C1 00 00 00 00 00 00 00 00 00 FF 01 00
9. C0 00 00 00 00 00 00 00 00 00 FF 11 00 C1 00 01
10. 00 00 00 00 00 00 00 FF 11 00 C1 00 00 00 00 00
11. 00 00 00 00 FF 01 00 C0 00 00 00 00 00 00 00 00
12. 00 FF 11 00 C1 00 01 00 00 00 00 00 00 00 E3 1B
13. 00 01 06 74 00 65 00 73 00 74 00 64 00 62 00 06
14. 6D 00 61 00 73 00 74 00 65 00 72 00 AB 66 00 45
15. 16 00 00 02 00 25 00 43 00 68 00 61 00 6E 00 67
16. 00 65 00 64 00 20 00 64 00 61 00 74 00 61 00 62
17. 00 61 00 73 00 65 00 20 00 63 00 6F 00 6E 00 74
18. 00 65 00 78 00 74 00 20 00 74 00 6F 00 20 00 27
19. 00 74 00 65 00 73 00 74 00 64 00 62 00 27 00 2E
20. 00 07 74 00 65 00 73 00 74 00 73 00 76 00 72 00
21. 00 01 00 00 00 E3 08 00 07 05 09 04 D0 00 34 00
22. E3 17 00 02 0A 75 00 73 00 5F 00 65 00 6E 00 67
23. 00 6C 00 69 00 73 00 68 00 00 AB 6A 00 47 16 00
24. 00 01 00 27 00 43 00 68 00 61 00 6E 00 67 00 65
25. 00 64 00 20 00 6C 00 61 00 6E 00 67 00 75 00 61
26. 00 67 00 65 00 20 00 73 00 65 00 74 00 74 00 69
27. 00 6E 00 67 00 20 00 74 00 6F 00 20 00 75 00 73
28. 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68
29. 00 2E 00 07 74 00 65 00 73 00 74 00 73 00 76 00
30. 72 00 00 01 00 00 00 AD 36 00 01 74 00 00 04 16
31. 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00
32. 74 00 20 00 53 00 51 00 4C 00 20 00 53 00 65 00
33. 72 00 76 00 65 00 72 00 00 00 00 00 0C 00 03 E8
34. E3 13 00 04 04 38 00 30 00 30 00 30 00 04 34 00
35. 30 00 39 00 36 00 AE 01 77 00 00 00 00 09 00 60
36. 81 14 FF E7 FF FF 00 02 02 07 01 04 01 00 05 04
37. FF FF FF FF 06 01 00 07 01 02 08 08 00 00 00 00
38. 00 00 00 00 09 04 FF FF FF FF 0B 47 35 00 44 00
39. 37 00 45 00 44 00 37 00 30 00 42 00 2D 00 42 00
40. 39 00 32 00 45 00 2D 00 34 00 31 00 32 00 42 00
41. 2D 00 42 00 33 00 32 00 46 00 2D 00 37 00 36 00
42. 30 00 43 00 44 00 37 00 34 00 44 00 42 00 39 00
43. 32 00 43 04 01 00 00 00 01 05 01 00 00 00 01 08
44. 01 00 00 00 01 FF FD 00 00 00 00 00 00 00 00 00
45. 00 00 00
46. <tds version="latest">
47. <PacketHeader>
48. <Type>
49. <BYTE>04 </BYTE>
50. </Type>
51. <Status>
52. <BYTE>01 </BYTE>
53. </Status>
54. <Length>
55. <BYTE>02 </BYTE>
56. <BYTE>C3 </BYTE>
57. </Length>
58. <SPID>
59. <BYTE>00 </BYTE>
60. <BYTE>77 </BYTE>
61. </SPID>
62. <Packet>
63. <BYTE>01 </BYTE>
64. </Packet>
65. <Window>
66. <BYTE>00 </BYTE>
67. </Window>
68. </PacketHeader>
69. <PacketData>
70. <TableResponse>
71. <DONEINPROC>
72. <TokenType>
73. <BYTE>FF </BYTE>
74. </TokenType>
75. <Status>
76. <USHORT>11 00 </USHORT>
77. </Status>
78. <CurCmd>
79. <USHORT>C1 00 </USHORT>
80. </CurCmd>
81. <DoneRowCount>
82. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
83. </DoneRowCount>
84. </DONEINPROC>
85. <DONEINPROC>
86. <TokenType>
87. <BYTE>FF </BYTE>
88. </TokenType>
89. <Status>
90. <USHORT>11 00 </USHORT>
91. </Status>
92. <CurCmd>
93. <USHORT>C1 00 </USHORT>
94. </CurCmd>
95. <DoneRowCount>
96. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
97. </DoneRowCount>
98. </DONEINPROC>
99. <DONEINPROC>
100. <TokenType>
101. <BYTE>FF </BYTE>
102. </TokenType>
103. <Status>
104. <USHORT>01 00 </USHORT>
105. </Status>
106. <CurCmd>
107. <USHORT>C0 00 </USHORT>
108. </CurCmd>
109. <DoneRowCount>
110. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
111. </DoneRowCount>
112. </DONEINPROC>
113. <DONEINPROC>
114. <TokenType>
115. <BYTE>FF </BYTE>
116. </TokenType>
117. <Status>
118. <USHORT>11 00 </USHORT>
119. </Status>
120. <CurCmd>
121. <USHORT>C1 00 </USHORT>
122. </CurCmd>
123. <DoneRowCount>
124. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
125. </DoneRowCount>
126. </DONEINPROC>
127. <DONEINPROC>
128. <TokenType>
129. <BYTE>FF </BYTE>
130. </TokenType>
131. <Status>
132. <USHORT>11 00 </USHORT>
133. </Status>
134. <CurCmd>
135. <USHORT>C1 00 </USHORT>
136. </CurCmd>
137. <DoneRowCount>
138. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
139. </DoneRowCount>
140. </DONEINPROC>
141. <DONEINPROC>
142. <TokenType>
143. <BYTE>FF </BYTE>
144. </TokenType>
145. <Status>
146. <USHORT>01 00 </USHORT>
147. </Status>
148. <CurCmd>
149. <USHORT>C0 00 </USHORT>
150. </CurCmd>
151. <DoneRowCount>
152. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
153. </DoneRowCount>
154. </DONEINPROC>
155. <DONEINPROC>
156. <TokenType>
157. <BYTE>FF </BYTE>
158. </TokenType>
159. <Status>
160. <USHORT>11 00 </USHORT>
161. </Status>
162. <CurCmd>
163. <USHORT>C1 00 </USHORT>
164. </CurCmd>
165. <DoneRowCount>
166. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
167. </DoneRowCount>
168. </DONEINPROC>
169. <DONEINPROC>
170. <TokenType>
171. <BYTE>FF </BYTE>
172. </TokenType>
173. <Status>
174. <USHORT>11 00 </USHORT>
175. </Status>
176. <CurCmd>
177. <USHORT>C1 00 </USHORT>
178. </CurCmd>
179. <DoneRowCount>
180. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
181. </DoneRowCount>
182. </DONEINPROC>
183. <DONEINPROC>
184. <TokenType>
185. <BYTE>FF </BYTE>
186. </TokenType>
187. <Status>
188. <USHORT>11 00 </USHORT>
189. </Status>
190. <CurCmd>
191. <USHORT>C1 00 </USHORT>
192. </CurCmd>
193. <DoneRowCount>
194. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
195. </DoneRowCount>
196. </DONEINPROC>
197. <DONEINPROC>
198. <TokenType>
199. <BYTE>FF </BYTE>
200. </TokenType>
201. <Status>
202. <USHORT>01 00 </USHORT>
203. </Status>
204. <CurCmd>
205. <USHORT>C0 00 </USHORT>
206. </CurCmd>
207. <DoneRowCount>
208. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
209. </DoneRowCount>
210. </DONEINPROC>
211. <DONEINPROC>
212. <TokenType>
213. <BYTE>FF </BYTE>
214. </TokenType>
215. <Status>
216. <USHORT>11 00 </USHORT>
217. </Status>
218. <CurCmd>
219. <USHORT>C1 00 </USHORT>
220. </CurCmd>
221. <DoneRowCount>
222. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
223. </DoneRowCount>
224. </DONEINPROC>
225. <DONEINPROC>
226. <TokenType>
227. <BYTE>FF </BYTE>
228. </TokenType>
229. <Status>
230. <USHORT>11 00 </USHORT>
231. </Status>
232. <CurCmd>
233. <USHORT>C1 00 </USHORT>
234. </CurCmd>
235. <DoneRowCount>
236. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
237. </DoneRowCount>
238. </DONEINPROC>
239. <DONEINPROC>
240. <TokenType>
241. <BYTE>FF </BYTE>
242. </TokenType>
243. <Status>
244. <USHORT>01 00 </USHORT>
245. </Status>
246. <CurCmd>
247. <USHORT>C0 00 </USHORT>
248. </CurCmd>
249. <DoneRowCount>
250. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
251. </DoneRowCount>
252. </DONEINPROC>
253. <DONEINPROC>
254. <TokenType>
255. <BYTE>FF </BYTE>
256. </TokenType>
257. <Status>
258. <USHORT>11 00 </USHORT>
259. </Status>
260. <CurCmd>
261. <USHORT>C1 00 </USHORT>
262. </CurCmd>
263. <DoneRowCount>
264. <LONGLONG>01 00 00 00 00 00 00 00 </LONGLONG>
265. </DoneRowCount>
266. </DONEINPROC>
267. <ENVCHANGE>
268. <BYTE>E3</BYTE>
269. <LENGTH>
270. 1B 00
271. </LENGTH>
272. <TOKEN type="database">
273. 01
274. </TOKEN>
275. <DATA>
276. 06 74 00 65 00 73 00 74 00 64 00 62 00 06 6D 00 61 00 73 00 74 00 65 00 72 00
277. </DATA>
278. </ENVCHANGE>
279. <INFO>
280. <BYTE>AB</BYTE>
281. <LENGTH>
282. 66 00
283. </LENGTH>
284. <DATA>
285. 45 16 00 00 02 00 25 00 43 00 68 00 61 00 6E 00
286. 67 00 65 00 64 00 20 00 64 00 61 00 74 00 61 00
287. 62 00 61 00 73 00 65 00 20 00 63 00 6F 00 6E 00
288. 74 00 65 00 78 00 74 00 20 00 74 00 6F 00 20 00
289. 27 00 74 00 65 00 73 00 74 00 64 00 62 00 27 00
290. 2E 00 07 74 00 65 00 73 00 74 00 73 00 76 00 72
291. 00 00 01 00 00 00
292. </DATA>
293. </INFO>
294. <ENVCHANGE>
295. <BYTE>E3</BYTE>
296. <LENGTH>
297. 08 00
298. </LENGTH>
299. <TOKEN type="sql collation">
300. 07
301. </TOKEN>
302. <DATA>
303. 05 09 04 D0 00 34 00
304. </DATA>
305. </ENVCHANGE>
306. <ENVCHANGE>
307. <BYTE>E3</BYTE>
308. <LENGTH>
309. 17 00
310. </LENGTH>
311. <TOKEN type="language">
312. 02
313. </TOKEN>
314. <DATA>
315. 0A 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00 73 00 68 00 00
316. </DATA>
317. </ENVCHANGE>
318. <INFO>
319. <BYTE>AB</BYTE>
320. <LENGTH>
321. 6A 00
322. </LENGTH>
323. <DATA>
324. 47 16 00 00 01 00 27 00 43 00 68 00 61 00 6E 00
325. 67 00 65 00 64 00 20 00 6C 00 61 00 6E 00 67 00
326. 75 00 61 00 67 00 65 00 20 00 73 00 65 00 74 00
327. 74 00 69 00 6E 00 67 00 20 00 74 00 6F 00 20 00
328. 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00
329. 73 00 68 00 2E 00 07 74 00 65 00 73 00 74 00 73
330. 00 76 00 72 00 00 01 00 00 00
331. </DATA>
332. </INFO>
333. <LOGINACK>
334. <BYTE>AD</BYTE>
335. <LENGTH>
336. 36 00
337. </LENGTH>
338. <INTERFACE>1</INTERFACE>
339. <TDSVERSION>74 00 00 04</TDSVERSION>
340. <PROGNAME>
341. 16
342. 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00
343. 74 00 20 00 53 00 51 00 4C 00 20 00 53 00 65 00
344. 72 00 76 00 65 00 72 00 00 00 00
345. </PROGNAME>
346. <VERSIONBUILD>
347. 0C 00 03 E8
348. </VERSIONBUILD>
349. <DATA>
350. 47 16 00 00 01 00 27 00 43 00 68 00 61 00 6E 00
351. 67 00 65 00 64 00 20 00 6C 00 61 00 6E 00 67 00
352. 75 00 61 00 67 00 65 00 20 00 73 00 65 00 74 00
353. 74 00 69 00 6E 00 67 00 20 00 74 00 6F 00 20 00
354. 75 00 73 00 5F 00 65 00 6E 00 67 00 6C 00 69 00
355. 73 00 68 00 2E 00 07 74 00 65 00 73 00 74 00 73
356. 00 76 00 72 00 00 01 00 00 00 00
357. </DATA>
358. </LOGINACK>
359. <ENVCHANGE>
360. <BYTE>E3</BYTE>
361. <LENGTH>
362. 13 00
363. </LENGTH>
364. <TOKEN type="packsize">
365. 04
366. </TOKEN>
367. <DATA>
368. 04 38 00 30 00 30 00 30 00 04 34 00 30 00 39 00 36 00
369. </DATA>
370. </ENVCHANGE>
371. <FeatureExtAck>
372. <BYTE>AE</BYTE>
373. <FeatureOpt>
374. <FeatureId>
375. <BYTE>01 </BYTE>
376. </FeatureId>
377. <FeatureDataLen>
378. <DWORD>77 00 00 00 </DWORD>
379. </FeatureDataLen>
380. <FeatureData>
381. <BYTE>
382. 00 09 00 60 81 14 FF E7 FF FF 00 02 02 07 01 04
383. 01 00 05 04 FF FF FF FF 06 01 00 07 01 02 08 08
384. 00 00 00 00 00 00 00 00 09 04 FF FF FF FF 0B 47
385. 35 00 44 00 37 00 45 00 44 00 37 00 30 00 42 00
386. 2D 00 42 00 39 00 32 00 45 00 2D 00 34 00 31 00
387. 32 00 42 00 2D 00 42 00 33 00 32 00 46 00 2D 00
388. 37 00 36 00 30 00 43 00 44 00 37 00 34 00 44 00
389. 42 00 39 00 32 00 43
390. </BYTE>
391. </FeatureData>
392. <FeatureId>
393. <BYTE>04 </BYTE>
394. </FeatureId>
395. <FeatureDataLen>
396. <DWORD>01 00 00 00 </DWORD>
397. </FeatureDataLen>
398. <FeatureData>
399. <BYTE>01</BYTE>
400. </FeatureData>
401. <FeatureId>
402. <BYTE>05 </BYTE>
403. </FeatureId>
404. <FeatureDataLen>
405. <DWORD>01 00 00 00 </DWORD>
406. </FeatureDataLen>
407. <FeatureData>
408. <BYTE>01</BYTE>
409. </FeatureData>
410. <FeatureId>
411. <AZURESQLSUPPORT>
412. <BYTE>08 </BYTE>
413. </AZURESQLSUPPORT>
414. </FeatureId>
415. <FeatureDataLen>
416. <DWORD>01 00 00 00 </DWORD>
417. </FeatureDataLen>
418. <FeatureData>
419. <BYTE>01</BYTE>
420. </FeatureData>
421. <TERMINATOR>
422. <BYTE>FF </BYTE>
423. </TERMINATOR>
424. </FeatureOpt>
425. </FeatureExtAck>
426. <DONE>
427. <TokenType>
428. <BYTE>FD </BYTE>
429. </TokenType>
430. <Status>
431. <USHORT>00 00 </USHORT>
432. </Status>
433. <CurCmd>
434. <USHORT>00 00 </USHORT>
435. </CurCmd>
436. <DoneRowCount>
437. <LONGLONG>00 00 00 00 00 00 00 00 </LONGLONG>
438. </DoneRowCount>
439. </DONE>
440. </TableResponse>
441. </PacketData>
442. </tds>

# Security

## 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, the exact choice of security mechanisms is determined by the SSPI layer. Likewise, although the decision as to whether channel encryption is 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 is used can 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.

## Index of Security Parameters

The following table lists the sections in this document in which the available Tabular Data Stream (TDS) security parameters are mentioned.

| Security parameter | Section |
| --- | --- |
| TLS Negotiation | [2.1](#Section_fd30432f71b2488cb30f19737d76d970) Transport  [3.2.5.1](#Section_cc823ca848674387819dcc5c19da5732) Sent Initial PRELOGIN Packet State  [3.2.5.2](#Section_d62e225bd8654ccc8f73de1ef49e30d4) Sent TLS/SSL Negotiation Packet State  [3.3.5.1](#Section_8bfcbcd21baf47928fff86a63a9e90fa) Initial State  [3.3.5.2](#Section_ef1c4791413f4ec7ad475810a514db94) TLS/SSL Negotiation State |
| SSPI Authentication | [2.2.1.2](#Section_f159ec89ef1a4fcd9ffd8ee701fccbb0) Login  [2.2.6.5](#Section_60f5640801884cd58b9025c6f2423868) PRELOGIN  [2.2.6.8](#Section_1dc6c197f606410badd9e6fe1dff0e8b) SSPI Message  [2.2.7.21](#Section_07e2bb7b8ba6445f89b1cc76d8bfa9c6) SSPI  3.2.5.1 Sent Initial PRELOGIN Packet State  3.2.5.2 Sent TLS/SSL Negotiation Packet State  [3.2.5.4](#Section_c373a90b113d4c5f826ad410365ee883) Sent LOGIN7 Record with SPNEGO Packet State  [3.3.5.3](#Section_1d82a68b57c246e984baed41f7bd0c7c) Login Ready State |
| Federated Authentication | 2.2.1.2 Login  [2.2.1.3](#Section_f40cca2929b843ecb9ff2f8682486c29) Federated Authentication Token  [2.2.2.3](#Section_1a22145b7657435d9c60318be82456e1) Federated Authentication Information  [2.2.6.3](#Section_827d963229574d54b9ea384530ae79d0) Federated Authentication Token  [2.2.6.4](#Section_773a62b6ee894c029e5e344882630aac) LOGIN7  2.2.6.5 PRELOGIN  [2.2.7.10](#Section_2eb82f8e11f046dcb42d27302fa4701a) FEATUREEXTACK  [2.2.7.11](#Section_0e4486d6d407496298030c1a4d4d87ce) FEDAUTHINFO  3.2.5.1 Sent Initial PRELOGIN Packet State  3.2.5.2 Sent TLS/SSL Negotiation Packet State  [3.2.5.3](#Section_ab7a745416a949ef8f208a0de2b31fbb) Sent LOGIN7 Record with Complete Authentication Token State  [3.2.5.5](#Section_c272e054279f4b09ac41f9ccf1d1ffde) Sent LOGIN7 Record with Federated Authentication Information Request State  3.3.5.1 Initial State  3.3.5.3 Login Ready State  [3.3.5.5](#Section_f6374988cc5446dbadbeb7742b8600e6) Federated Authentication Ready State |

# Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

This document specifies version-specific details in the Microsoft .NET Framework. For information about which versions of .NET Framework are available in each released Windows product or as supplemental software, see [[MS-NETOD]](%5bMS-NETOD%5d.pdf#Section_bcca8164da0843f2a983c34ed99171b0) section 4.

* Microsoft .NET Framework 1.1
* Microsoft .NET Framework 2.0
* Microsoft .NET Framework 4.0
* Microsoft .NET Framework 4.5
* Microsoft .NET Framework 4.6
* Microsoft .NET Framework 4.7
* Microsoft .NET Framework 4.8
* 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
* Microsoft SQL Server 2016
* Microsoft SQL Server 2017

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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](#Appendix_A_Target_1): The following table describes the latest TDS version that is supported by a particular version of Microsoft SQL Server. To determine the earliest TDS version that is supported by a particular SQL Server version, refer to the product documentation.

| TDS version | SQL Server version |
| --- | --- |
| 7.0 | SQL Server 7.0 |
| 7.1 | SQL Server 2000 |
| 7.1 Revision 1 | SQL Server 2000 SP1 |
| 7.2 | SQL Server 2005 |
| 7.3.A | SQL Server 2008 |
| 7.3.B | SQL Server 2008 R2 |
| 7.4 | SQL Server 2012  SQL Server 2014  SQL Server 2016  SQL Server 2017 |

The following table describes the TDS versions that are supported by particular versions of the .NET Framework.

| TDS version | .NET Framework version |
| --- | --- |
| 7.0 | .NET Framework 1.1 |
| 7.1 | .NET Framework 1.1 |
| 7.1 Revision 1 | .NET Framework 1.1 |
| 7.2 | .NET Framework 2.0 |
| 7.3.A | .NET Framework 2.0  .NET Framework 4.0 |
| 7.3.B | .NET Framework 2.0  .NET Framework 4.0 |
| 7.4 | .NET Framework 4.5  .NET Framework 4.6  .NET Framework 4.7  .NET Framework 4.8 |

[<2> Section 2.1](#Appendix_A_Target_2): Microsoft Windows Named Pipes in message mode [[MSDN-NP]](https://go.microsoft.com/fwlink/?LinkId=90247). Please see [[MSDN-NamedPipes]](https://go.microsoft.com/fwlink/?LinkId=127839) for additional information related to Microsoft-specific implementations.

[<3> Section 2.1](#Appendix_A_Target_3): VIA is supported only by SQL Server 7.0, SQL Server 2000, SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2. This means that VIA will never be the underlying transport protocol if either the server or the client can support TDS version 7.4.

[<4> Section 2.2.1.3](#Appendix_A_Target_4): Federated authentication is not supported by SQL Server.

[<5> Section 2.2.3.1.1](#Appendix_A_Target_5): Only legacy clients that support SQL Server versions that were released prior to SQL Server 7.0 can use Pre-TDS7 Login.

[<6> Section 2.2.3.1.1](#Appendix_A_Target_6): Only clients that support SQL Server 7.0 or later can use TDS7 Login.

[<7> Section 2.2.3.1.5](#Appendix_A_Target_7): Depending on the message type and provider, such as Microsoft SQL Server Native Client or Microsoft .NET Framework Data Provider for SQL Server, PacketID values start with either 0 or 1, which is an implementation choice. The .NET Framework Data Provider for SQL Server uses 1.

[<8> Section 2.2.4.3](#Appendix_A_Target_8): 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**](#gt_26c1caf3-c889-4b99-a22b-9da056d397cf) write.

[<9> Section 2.2.5.1.2](#Appendix_A_Target_9): The sorting styles that are used by SQL Server are described in [[MSDN-ColSortSty]](https://go.microsoft.com/fwlink/?LinkId=233328).

[<10> Section 2.2.5.1.2](#Appendix_A_Target_10): COLLATION represents a collation in SQL Server, as described in [[MSDN-Collation]](https://go.microsoft.com/fwlink/?LinkId=233327). It can be either a SQL Server collation or a Windows collation.

Version can be of value 0, 1, 2, or 3. A value of 0 denotes collations introduced in SQL Server 2000. A value of 1 denotes collations introduced in SQL Server 2005. A value of 2 denotes collations introduced in SQL Server 2008. A value of 3 denotes collations introduced in SQL Server 2017.

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**](#gt_c305d0ab-8b94-461a-bd76-13b40cb8c4d8) data, the SortId is used to derive comparison information flags, such as whether, for a given SortId, a lowercase "a" equals an uppercase "A".

[<11> Section 2.2.5.3.1](#Appendix_A_Target_11): Query notifications is not supported by SQL Server 7.0 and SQL Server 2000.

[<12> Section 2.2.5.3.1](#Appendix_A_Target_12): SSBDeployment corresponds to the SQL Server Service Broker deployment version.

[<13> Section 2.2.5.4.1](#Appendix_A_Target_13): NULLTYPE can be sent to SQL Server (for example, in RPCRequest), but SQL Server never emits NULLTYPE data.

[<14> Section 2.2.5.5.3](#Appendix_A_Target_14): When a .NET Framework Data Provider for SQL Server accesses an XML field, the returned data value is encoded in binary XML format [[MS-BINXML]](%5bMS-BINXML%5d.pdf#Section_11ab6e8d247244d1a9e6bddf000e12f6). For other providers, the value is sent in Unicode text format.

[<15> Section 2.2.5.5.4](#Appendix_A_Target_15): Microsoft implementations return an error if a client does send a raw collation within a sql\_variant.

[<16> Section 2.2.6.3](#Appendix_A_Target_16): Federated Authentication and the FEDAUTH token are not supported by SQL Server.

[<17> Section 2.2.6.4](#Appendix_A_Target_17): 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  SQL Server 2016  SQL Server 2017 | 0x04000074 |

[<18> Section 2.2.6.4](#Appendix_A_Target_18): The value "1" for fByteOrder is supported only by SQL Server 7.0.

[<19> Section 2.2.6.4](#Appendix_A_Target_19): SQL Server assumes fFloat to be FLOAT\_IEEE\_754 and ignores the other settings.

[<20> Section 2.2.6.4](#Appendix_A_Target_20): For fODBC, SQL Server returns a value of zero for ROWCOUNT.

[<21> Section 2.2.6.4](#Appendix_A_Target_21): For fOLEDB, SQL Server returns a value of zero for ROWCOUNT.

[<22> Section 2.2.6.4](#Appendix_A_Target_22): SQL Server implementations do not inspect the fSendYukonBinaryXML bit. When using the .NET Framework Data Provider for SQL Server, the server sends binary XML if the TDS version is 7.2 or later.

[<23> Section 2.2.6.4](#Appendix_A_Target_23): The FEDAUTH feature extension is not supported by SQL Server.

[<24> Section 2.2.6.4](#Appendix_A_Target_24): The COLUMNENCRYPTION feature extension is not supported by SQL Server 7.0, SQL Server 2000, SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, and SQL Server 2014.

[<25> Section 2.2.6.4](#Appendix_A_Target_25): Enclave computations are not supported by SQL Server. Support for this functionality was introduced in the .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<26> Section 2.2.6.4](#Appendix_A_Target_26): The **EnclaveType** field is not supported by SQL Server. This field was introduced in the .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<27> Section 2.2.6.4](#Appendix_A_Target_27): The GLOBALTRANSACTIONS feature extension is not supported by SQL Server.

[<28> Section 2.2.6.4](#Appendix_A_Target_28): The AZURESQLSUPPORT feature extension is not supported by SQL Server. This feature extension was introduced in .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<29> Section 2.2.6.5](#Appendix_A_Target_29): The FEDAUTHREQUIRED payload option token is not supported by SQL Server.

[<30> Section 2.2.6.5](#Appendix_A_Target_30): In SQL Server 2012, SQL Server 2014, and SQL Server 2016, 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" can 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.

[<31> Section 2.2.6.6](#Appendix_A_Target_31): The fNoMetaData flag is not supported by SQL Server 2016.

[<32> Section 2.2.6.6](#Appendix_A_Target_32): The EnclavePackage parameter is not supported by SQL Server. This parameter was introduced in the .NET Framework 4.7.2 and is not supported by .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<33> Section 2.2.7.1](#Appendix_A_Target_33): ALTMETADATA\_TOKEN is supported only by SQL Server 7.0, SQL Server 2000, SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2.

[<34> Section 2.2.7.2](#Appendix_A_Target_34): ALTROW\_TOKEN is supported only by SQL Server 7.0, SQL Server 2000, SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2.

[<35> Section 2.2.7.4](#Appendix_A_Target_35): SQL Server 2016 supports the fHidden flag only through a many-to-many result and by connecting via ODBC.

[<36> Section 2.2.7.4](#Appendix_A_Target_36): The NoMetaData parameter is not supported by SQL Server 2016.

[<37> Section 2.2.7.5](#Appendix_A_Target_37): The 0x4: DONE\_INXACT bit is not set by SQL Server and is reserved for future use.

[<38> Section 2.2.7.5](#Appendix_A_Target_38): The [DONE](#Section_3c06f11098bd4d5bb836b1ba66452cb7) 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.

When login fails for any reason, SQL Server might also send an error message followed by a DONE token. In this case, the server has already completed 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.

[**SQL Server Native Client (SNAC)**](#gt_f5b7b5b5-50e8-4f63-8e66-ad8a30b229f2) [[MSDN-SNAC]](https://go.microsoft.com/fwlink/?LinkId=213738) and SQLClient use the VERSION option in the Pre-Login Response message to detect whether DoneRowCount is LONG or ULONGLONG. It is ULONGLONG if VERSION in the Pre-Login Response message indicates that the server is SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, SQL Server 2014, or SQL Server 2016. Otherwise, DoneRowCount is LONG.

A third-party implementation has 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.

[<39> Section 2.2.7.6](#Appendix_A_Target_39): The 0x4: DONE\_INXACT bit is not set by SQL Server and is reserved for future use.

[<40> Section 2.2.7.7](#Appendix_A_Target_40): The 0x4: DONE\_INXACT bit is not set by SQL Server and is reserved for future use.

[<41> Section 2.2.7.8](#Appendix_A_Target_41): Type 16: Transaction Manager Address is not used by SQL Server.

[<42> Section 2.2.7.9](#Appendix_A_Target_42): Numbers less than 20001 are reserved by SQL Server.

[<43> Section 2.2.7.9](#Appendix_A_Target_43): SQL Server does not raise system errors with severities of 0 through 9.

[<44> Section 2.2.7.9](#Appendix_A_Target_44): For compatibility reasons, SQL Server converts severity 10 to severity 0 before returning the error information to the calling application.

[<45> Section 2.2.7.10](#Appendix_A_Target_45): The FEDAUTH feature extension is not supported by SQL Server.

[<46> Section 2.2.7.10](#Appendix_A_Target_46): The COLUMNENCRYPTION feature extension is not supported by SQL Server 7.0, SQL Server 2000, SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, and SQL Server 2014.

[<47> Section 2.2.7.10](#Appendix_A_Target_47): Enclave computations are not supported by SQL Server. Support for this feature was introduced in the .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<48> Section 2.2.7.10](#Appendix_A_Target_48): The **EnclaveType** field is not supported by SQL Server. This field was introduced in the .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<49> Section 2.2.7.10](#Appendix_A_Target_49): The GLOBALTRANSACTIONS feature extension is not supported by SQL Server.

[<50> Section 2.2.7.10](#Appendix_A_Target_50): The AZURESQLSUPPORT feature extension is not supported by SQL Server. This feature extension was introduced in .NET Framework 4.7.2 and is not supported by the .NET Framework 1.1, .NET Framework 2.0, .NET Framework 4.0, .NET Framework 4.5, .NET Framework 4.6, .NET Framework 4.7, and .NET Framework 4.7.1.

[<51> Section 2.2.7.11](#Appendix_A_Target_51): The FEDAUTHINFO token is not supported by SQL Server.

[<52> Section 2.2.7.12](#Appendix_A_Target_52): Numbers less than 20001 are reserved by SQL Server.

[<53> Section 2.2.7.13](#Appendix_A_Target_53): 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  SQL Server 2014  SQL Server 2016  SQL Server 2017 | 0x04000074 | 0x74000004 |

\*SQL Server 2008 TDS version 0x03000A73 does not include support for NBCROW and fSparseColumnSet.

[<54> Section 3.2.2](#Appendix_A_Target_54): In Microsoft implementations, the default value for the [**Microsoft/Windows Data Access Components (MDAC/WDAC)**](#gt_4f587fbb-99d6-4d8c-aebd-b4b325ce64d8) 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]](https://go.microsoft.com/fwlink/?LinkId=213737).

[<55> Section 3.2.2](#Appendix_A_Target_55): 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].

# Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

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.
* A document revision that captures changes to protocol functionality.

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 **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact [dochelp@microsoft.com](mailto:dochelp@microsoft.com).

| Section | Description | Revision class |
| --- | --- | --- |
| [1.3](#Section_893fcc7e8a394b3c815a773b7b982c50) Overview | Added .NET Framework 4.8 to the product behavior note. | Major |
| [2](#Section_8d0d4f5624a145ffb0bacc7d17517e82) Messages | Clarified that the underscore ("\_") is a valid ABNF character within an identifier and that "%x00" and "%b0" are valid values. | Minor |
| [6](#Section_135d0ebe5c4c4a9499bf1811eccb9f4a) Appendix A: Product Behavior | Added .NET Framework 4.8 to the list of applicable products. | Major |

# Index

A

Abstract data model

client ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.2.1](#section_0e7007a13da74323914828d89ec2ac48) 120)

server ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.3.1](#section_1457f2976d7d4b47a4bef35d273b8cae) 128)

ALL\_HEADERS rule definition

[overview](#section_e17e54ae0fac48b7b8a8c267be297923) 33

[Query Notifications header](#section_e168d373a7b741aab6ca25985466a7e0) 34

[Transaction Descriptor header](#section_4257dd95ef6c4621b75d270738487d68) 34

[Applicability](#section_6e698116b1f3467889c33ca65cfdd8ed) 15

[Attention message](#section_dc28579f49b14a789c5f63fbda002d2e) 19

[Attention request example](#section_fa485de40e10470494d24d2df737b29e) 158

[Attention tokens](#section_ef2ba0a16e4b46d09aceb6cc849e417a) 27

C

[Capability negotiation](#section_6809de85166544ada4596f6621e02257) 15

[Change tracking](#section_47286277defd46f3a074e769bf9dab1e) 199

Client

abstract data model ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.2.1](#section_0e7007a13da74323914828d89ec2ac48) 120)

higher-layer triggered events ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.2.4](#section_e7af6058672941438815f82a475f9ba6) 122)

initialization ([section 3.1.3](#section_07dcb45cdb9c4ce981cd02b370cb4809) 114, [section 3.2.3](#section_73aec80675674e189d1bf8e8cac78522) 121)

[local events](#section_2ceaed83ecdf479eba465fdcbd888ddc) 119

message processing ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.2.5](#section_bd3a16dd75b64546933059c3ef44d50e) 123)

[Final state](#section_07a0c71e1a7b44558da47e43a09589bc) 127

[Logged In state](#section_e2e7293f55cf43eeb0282d6dea625db6) 126

[Sent Attention state](#section_61fb4f9d95ec41718d4bdac4eb16149f) 126

[Sent Client Request state](#section_c89701263cbc446f9275cc36ba62a2f4) 126

[Sent Initial PRELOGIN Packet state](#section_cc823ca848674387819dcc5c19da5732) 123

[Sent LOGIN7 Record with SPNEGO Packet state](#section_c373a90b113d4c5f826ad410365ee883) 125

[Sent LOGIN7 Record with Standard Login state](#section_ab7a745416a949ef8f208a0de2b31fbb) 125

[Sent TLS/SSL Negotiation Packet state](#section_d62e225bd8654ccc8f73de1ef49e30d4) 124

messages

[Attention](#section_dc28579f49b14a789c5f63fbda002d2e) 19

[login](#section_f159ec89ef1a4fcd9ffd8ee701fccbb0) 18

[overview](#section_7ea9ee1ab46141f29004141c0e712935) 17

[pre-login](#section_58886b79ec7542f3bd7be32f327366b6) 18

[remote procedure call](#section_26327437aa3c4e969bba73a6e862ba21) 19

[SQL command](#section_5b7f9062d5144b4ea7c59a27a19596c4) 18

[SQL command with binary data](#section_88176081df754b24bcfb4c16ff03cbfa) 18

[transaction manager request](#section_b4b7856454404fc0b5efc9e1925aaefe) 19

[other local events](#section_97d920799ea14a7ba915fa7e8045d88b) 127

overview ([section 3.1](#section_34e6da7dd80c4875b1553be341822623) 114, [section 3.2](#section_ce1cef02f43c4e37a190aaa444b96360) 120)

sequencing rules ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.2.5](#section_bd3a16dd75b64546933059c3ef44d50e) 123)

[Final state](#section_07a0c71e1a7b44558da47e43a09589bc) 127

[Logged In state](#section_e2e7293f55cf43eeb0282d6dea625db6) 126

[Sent Attention state](#section_61fb4f9d95ec41718d4bdac4eb16149f) 126

[Sent Client Request state](#section_c89701263cbc446f9275cc36ba62a2f4) 126

[Sent Initial PRELOGIN Packet state](#section_cc823ca848674387819dcc5c19da5732) 123

[Sent LOGIN7 Record with SPNEGO Packet state](#section_c373a90b113d4c5f826ad410365ee883) 125

[Sent LOGIN7 Record with Standard Login state](#section_ab7a745416a949ef8f208a0de2b31fbb) 125

[Sent TLS/SSL Negotiation Packet state](#section_d62e225bd8654ccc8f73de1ef49e30d4) 124

timer events ([section 3.1.6](#section_9d65081e87bd4461a6e4ff4bb4895b3c) 119, [section 3.2.6](#section_030af85119fd4a9ca7df05b637f7d888) 127)

timers ([section 3.1.2](#section_2192569232bd484d974bd31e025becc8) 114, [section 3.2.2](#section_8313b90eeee347e1844173b3b5df1e39) 121)

[Client Messages message](#section_7ea9ee1ab46141f29004141c0e712935) 17

[Client Request Execution state](#section_21c9c608c6bc456fb7eb9f09a90bd282) 133

D

Data model - abstract

client ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.2.1](#section_0e7007a13da74323914828d89ec2ac48) 120)

server ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.3.1](#section_1457f2976d7d4b47a4bef35d273b8cae) 128)

Data stream types

[data type dependent data streams](#section_3f983fde0509485a8c40a9fa6679a828) 31

[unknown-length data streams](#section_1893c1ffedce4465b194022039c98928) 31

[variable-length data streams](#section_47acf60e9c364184b016a4947263e25f) 31

Data type definitions

[fixed-length data types](#section_859eb3d280d340f6a637414552c9c552) 35

[overview](#section_ffb02215af074b5085451fd522106c68) 35

[partially length-prefixed data types](#section_7d26a257083e409b81ba897e0c672be0) 39

[SQL\_VARIANT](#section_2435e85d9e61492cacb2627ffccb5b92) 42

Table Valued Parameter

[metadata](#section_0dfc5367a3884c929ba44d28e775acbc) 43

[optional metadata tokens](#section_fcacd8f20bf04118809f8d460c4e1508) 45

[overview](#section_c264db71c1ec4fe8b5ef19d54b1e6566) 43

[TDS type restrictions](#section_1dd985a98dbb44e882d4ed8ab7671ef3) 47

[UDT Assembly Information](#section_a57df60ed0a64e7ea2e5ccacd277c673) 41

[variable-length data types](#section_ce3183a69d8947e8a02fde5a1a1303de) 36

[XML data type](#section_bfebeca7a3a643dea7e7fa721e803b4e) 42

[DONE tokens](#section_ef2ba0a16e4b46d09aceb6cc849e417a) 27

E

[Error messages](#section_7fffdf0c78d140fd9dee1e6b4246d3aa) 21

Examples

[attention request](#section_fa485de40e10470494d24d2df737b29e) 158

[FeatureExt with AZURESQLSUPPORT Feature Data](#section_81d084b0ea234a9cbe4e7aadbc5a88c3) 181

[FeatureExt with SESSIONRECOVERY feature data](#section_b238ee1d249d478086a8a777f66623f2) 168

[FeatureExtAck with AZURESQLSUPPORT Feature Data](#section_f14d26d6212a42e499d79bc20ccefba2) 184

[FeatureExtAck with SESSIONRECOVERY feature data](#section_ef5774d818a543aebedc316e75e2618b) 173

[login request](#section_ce5ad23f6bf84fa594266b0d36e14da2) 136

[login request with federated authentication](#section_f88b63bbb47949e1a87bdeda521da508) 138

[login response](#section_517a62a2744847b681ebc0c5027826ca) 145

[login response with federated authentication](#section_1582e9753662411e9b27c23cc70a8b4b) 148

[overview](#section_e48ebc472e954970a116da45862af90b) 135

[pre-login request](#section_9420b4a3eb9f4f5e90bd3160444aa5a7) 135

[RPC client request](#section_1469b2fa6cab42e991f9044d358b306b) 155

[RPC server response](#section_a4e706cf8ae7480987a1ce293a6f7f68) 157

[SparseColumn select statement](#section_b08d4f24b4bd4b5ebda7d74071ae56a1) 163

[SQL batch client request](#section_b05b006b3cbf404bbcaf7ec584b54706) 152

[SQL batch server response](#section_d2a06235fbef4223b630a90dea24a3d7) 153

[SQL command with binary data](#section_43dac55aa31a4122ac0620cfedf76811) 159

[SSPI message](#section_dc57840ad13b43a1aad35425afadbc0c) 158

[Table response with SESSIONSTATE token data](#section_f3a3d7443b7c4b7792fdd61bca047ef0) 178

[token data stream](#section_ee3cc0fa23474a10b22f9e316470e826) 180

[attention signal - out-of-band](#section_f8aa004ebf764f94bd5b39597a888146) 180

[sending an SQL batch](#section_0d27294c86d5400d897faeedca6c4e13) 180

[transaction manager request](#section_9f10990a9c744a3989da2a3b05effacb) 160

[TVP insert statement](#section_2821fec5e1f642d281e840a2b7fead2d) 161

F

[FeatureExt with AZURESQLSUPPORT Feature Data example](#section_81d084b0ea234a9cbe4e7aadbc5a88c3) 181

[FeatureExt with SESSIONRECOVERY feature data example](#section_b238ee1d249d478086a8a777f66623f2) 168

[FeatureExtAck with AZURESQLSUPPORT Feature Data example](#section_f14d26d6212a42e499d79bc20ccefba2) 184

[FeatureExtAck with SESSIONRECOVERY feature data example](#section_ef5774d818a543aebedc316e75e2618b) 173

[Fields - vendor-extensible](#section_f3a6041cfe84420abda941021dee1cbe) 16

Final state ([section 3.2.5.10](#section_07a0c71e1a7b44558da47e43a09589bc) 127, [section 3.3.5.9](#section_f9157109df6b4f23ab9135a5ca43260d) 134)

[Fixed-length token](#section_5d73136323384ebeb1eb5cce0780ef8f) 26

G

[Glossary](#section_26cd55fe51c44948bd6b552f861e9427) 8

Grammar definition - token description

[data packet stream tokens](#section_f79bb5b85919439aa69648064b78b091) 50

data stream types

[data type dependent data streams](#section_3f983fde0509485a8c40a9fa6679a828) 31

[unknown-length data streams](#section_1893c1ffedce4465b194022039c98928) 31

[variable-length data streams](#section_47acf60e9c364184b016a4947263e25f) 31

data type definitions

[fixed-length data types](#section_859eb3d280d340f6a637414552c9c552) 35

[overview](#section_ffb02215af074b5085451fd522106c68) 35

[partially length-prefixed data types](#section_7d26a257083e409b81ba897e0c672be0) 39

[SQL\_VARIANT](#section_2435e85d9e61492cacb2627ffccb5b92) 42

[Table Valued Parameter](#section_c264db71c1ec4fe8b5ef19d54b1e6566) 43

[UDT Assembly Information](#section_a57df60ed0a64e7ea2e5ccacd277c673) 41

[variable-length data types](#section_ce3183a69d8947e8a02fde5a1a1303de) 36

[XML data type](#section_bfebeca7a3a643dea7e7fa721e803b4e) 42

general rules

[collation rule definition](#section_3d29e8dc218a42c69ba4947ebca9fd7e) 30

[least significant bit order](#section_bbc22f15e1a04338a169c79819d39b1c) 30

[overview](#section_d2ed21d6527b46ac803594f6f68eb9a8) 27

packet data stream headers

[overview](#section_e17e54ae0fac48b7b8a8c267be297923) 33

[Query Notifications header](#section_e168d373a7b741aab6ca25985466a7e0) 34

[Transaction Descriptor header](#section_4257dd95ef6c4621b75d270738487d68) 34

[TYPE\_INFO rule definition](#section_cbe9c510eae64b1f9893a098944d430a) 48

[Grammar Definition for Token Description message](#section_b7b3f6ba65dc491eb84ddbb12654228d) 27

H

Higher-layer triggered events

client ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.2.4](#section_e7af6058672941438815f82a475f9ba6) 122)

server ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.3.4](#section_6af826c5a9574438a7c0b3bbd309c69a) 129)

I

[Implementer - security considerations](#section_5d627db5467d40249aa8da067f419096) 192

[Index of security parameters](#section_30acd29386b242338307d02ac3b7d55c) 192

[Informational messages](#section_7fffdf0c78d140fd9dee1e6b4246d3aa) 21

[Informative references](#section_5e29795df0ab4f25b1f5f65bea469f28) 11

[Initial state](#section_8bfcbcd21baf47928fff86a63a9e90fa) 129

Initialization

client ([section 3.1.3](#section_07dcb45cdb9c4ce981cd02b370cb4809) 114, [section 3.2.3](#section_73aec80675674e189d1bf8e8cac78522) 121)

server ([section 3.1.3](#section_07dcb45cdb9c4ce981cd02b370cb4809) 114, [section 3.3.3](#section_d53254d96e6f4339aef09439e2269688) 129)

[Introduction](#section_1ef08b76159440cf8ce0d2407133dd3d) 8

L

Local events

client ([section 3.1.7](#section_2ceaed83ecdf479eba465fdcbd888ddc) 119, [section 3.2.7](#section_97d920799ea14a7ba915fa7e8045d88b) 127)

[server](#section_2ceaed83ecdf479eba465fdcbd888ddc) 119

Logged In state ([section 3.2.5.6](#section_e2e7293f55cf43eeb0282d6dea625db6) 126, [section 3.3.5.6](#section_7c37e9c36c6544dfbbd81e8fc11a618d) 133)

[Login Ready state](#section_1d82a68b57c246e984baed41f7bd0c7c) 130

[Login request example](#section_ce5ad23f6bf84fa594266b0d36e14da2) 136

[Login request with federated authentication example](#section_f88b63bbb47949e1a87bdeda521da508) 138

[Login response example](#section_517a62a2744847b681ebc0c5027826ca) 145

[Login response with federated authentication example](#section_1582e9753662411e9b27c23cc70a8b4b) 148

M

Message processing

client ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.2.5](#section_bd3a16dd75b64546933059c3ef44d50e) 123)

[Final state](#section_07a0c71e1a7b44558da47e43a09589bc) 127

[Logged In state](#section_e2e7293f55cf43eeb0282d6dea625db6) 126

[Sent Attention state](#section_61fb4f9d95ec41718d4bdac4eb16149f) 126

[Sent Client Request state](#section_c89701263cbc446f9275cc36ba62a2f4) 126

[Sent Initial PRELOGIN Packet state](#section_cc823ca848674387819dcc5c19da5732) 123

[Sent LOGIN7 Record with SPNEGO Packet state](#section_c373a90b113d4c5f826ad410365ee883) 125

[Sent LOGIN7 Record with Standard Login state](#section_ab7a745416a949ef8f208a0de2b31fbb) 125

[Sent TLS/SSL Negotiation Packet state](#section_d62e225bd8654ccc8f73de1ef49e30d4) 124

server ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.3.5](#section_8d3c50da023f4bc5959c60f38ed22825) 129)

[Client Request Execution state](#section_21c9c608c6bc456fb7eb9f09a90bd282) 133

[Final state](#section_f9157109df6b4f23ab9135a5ca43260d) 134

[Initial state](#section_8bfcbcd21baf47928fff86a63a9e90fa) 129

[Logged In state](#section_7c37e9c36c6544dfbbd81e8fc11a618d) 133

[Login Ready state](#section_1d82a68b57c246e984baed41f7bd0c7c) 130

[SPNEGO Negotiation state](#section_7b617d06e13845f3bcea35619b14ac72) 132

[TLS/SSL Negotiation state](#section_ef1c4791413f4ec7ad475810a514db94) 130

Messages

[Client Messages](#section_7ea9ee1ab46141f29004141c0e712935) 17

[Grammar Definition for Token Description](#section_b7b3f6ba65dc491eb84ddbb12654228d) 27

[overview](#section_8d0d4f5624a145ffb0bacc7d17517e82) 17

[Packet Data Token and Tokenless Data Streams](#section_dc3a08548230482fbbb9d94a3b905a26) 25

[Packet Data Token Stream Definition](#section_67b6113cd72242d1902c3f6e8de09173) 77

[Packets](#section_e5ea85201ea34a75a2a9c17e63e9ee19) 21

[Server Messages](#section_342f4cbb2b4b489c8b63f99b12021a94) 19

syntax

[client messages](#section_7ea9ee1ab46141f29004141c0e712935) 17

[grammar definition for token description](#section_b7b3f6ba65dc491eb84ddbb12654228d) 27

[overview](#section_a734db771cb247e4b61a0c9dbd8f960f) 17

[packet data token and tokenless data streams](#section_dc3a08548230482fbbb9d94a3b905a26) 25

[packet data token stream definition](#section_67b6113cd72242d1902c3f6e8de09173) 77

[packet header message type - stream definition](#section_c060af9c6db74360954cc28a132c8949) 50

[packets](#section_e5ea85201ea34a75a2a9c17e63e9ee19) 21

[server messages](#section_342f4cbb2b4b489c8b63f99b12021a94) 19

[transport](#section_fd30432f71b2488cb30f19737d76d970) 17

N

[Normative references](#section_559375dcef824a2ca36ec5ec40c83f9d) 10

O

Other local events

[client](#section_97d920799ea14a7ba915fa7e8045d88b) 127

[server](#section_388d89cc00a34f57bc22624e251e68f3) 134

[Overview (synopsis)](#section_893fcc7e8a394b3c815a773b7b982c50) 13

P

Packet data - token stream definition

[ALTMETADATA](#section_004bba4a8c234d7bab2cd9e7ba864cd0) 78

[ALTROW](#section_d1c427616a6443ab8a55fccb210ac073) 80

[COLINFO](#section_aa8466c5ca3d48caa6387c1becebe754) 81

[COLMETADATA](#section_58880b9f381c43b2bf8b0727a98c4f4c) 82

[DONE](#section_3c06f11098bd4d5bb836b1ba66452cb7) 85

[DONEINPROC](#section_43e891c5f7a1432f8f9f233c4cd96afb) 87

[DONEPROC](#section_65e24140edea46e5b710209af2016195) 88

[ENVCHANGE](#section_2b3eb7e5d43d4d1bbf4d76b9e3afc791) 89

[ERROR](#section_9805e9fa1f8b4cf88f788d2602228635) 93

[FEATUREEXTACK](#section_2eb82f8e11f046dcb42d27302fa4701a) 95

[INFO](#section_284bb815d0834ed5b33abdc2492e322b) 100

[LOGINACK](#section_490e563dcc6e4c86bb95ef0186b98032) 101

[NBCROW](#section_c0da3cc94441476baa995a518f22dc0a) 102

[OFFSET](#section_8d0b37ff20c1439e8f311d7f136249b5) 104

[ORDER](#section_252759be9d744435809dd55dd860ea78) 104

[RETURNSTATUS](#section_c719f199e71b418790b994f78bd1870e) 105

[RETURNVALUE](#section_7091f6f6b83d4ed2afebba5013dfb18f) 106

[ROW](#section_3840ef933b104aca9fd1a210b8bb6d0c) 108

[SESSIONSTATE](#section_626fbe19f3564599ba17c70f44005106) 109

[SSPI](#section_07e2bb7b8ba6445f89b1cc76d8bfa9c6) 111

[Table Valued Parameter row](#section_103eb1d15514418296ece95896904f50) 112

[TABNAME](#section_140e3348da08409ab6c3f0fc9cee2d6e) 111

Packet data stream headers

[overview](#section_e17e54ae0fac48b7b8a8c267be297923) 33

[Query Notifications header](#section_e168d373a7b741aab6ca25985466a7e0) 34

[Transaction Descriptor header](#section_4257dd95ef6c4621b75d270738487d68) 34

[Packet Data Token and Tokenless Data Streams message](#section_dc3a08548230482fbbb9d94a3b905a26) 25

[Packet Data Token Stream Definition message](#section_67b6113cd72242d1902c3f6e8de09173) 77

Packet header message type - stream definition

[BulkLoad - UpdateText/WriteText](#section_6ee5bed798d844f992177b10e2123596) 51

[BulkLoadBCP](#section_ab4a7d62cd1f4db1b67decae58f493e3) 50

[LOGIN7](#section_773a62b6ee894c029e5e344882630aac) 53

[PRELOGIN](#section_60f5640801884cd58b9025c6f2423868) 65

[RPCRequest](#section_619c43b694954a589e49a4950db245b3) 69

[SQLBatch](#section_f2026cd39a464a3f9a08f63140bcbbe3) 73

[SSPI message](#section_1dc6c197f606410badd9e6fe1dff0e8b) 73

[transaction manager request](#section_0fb28ba5ddcb4d0295c3aa5b05ec6092) 74

Packets

[overview](#section_e5ea85201ea34a75a2a9c17e63e9ee19) 21

[packet data](#section_3420734b287b4cdea0dd538c770a89a1) 24

packet header

[Length](#section_c1cddd03b448470a946a9b1b908f27a7) 24

[overview](#section_7af536671b72470382587984e838f746) 22

[PacketID](#section_ec9e8663191c4dd1baa848bbfba5ed7e) 24

[SPID](#section_fcfc00d06df142c88d3493007b9a80f0) 24

[Status](#section_ce398f9a7d474ede8f369dd6fc21ca43) 23

[Type](#section_9b4a463c26344a4bac35bebfff2fb0f7) 22

[Window](#section_fbc2f52392e64316aaed3a6966e548ad) 24

[Packets message](#section_e5ea85201ea34a75a2a9c17e63e9ee19) 21

[Parameters - security index](#section_30acd29386b242338307d02ac3b7d55c) 192

[Preconditions](#section_58fb5c727510416e81109180ede8628c) 14

[Pre-login request example](#section_9420b4a3eb9f4f5e90bd3160444aa5a7) 135

[Prerequisites](#section_58fb5c727510416e81109180ede8628c) 14

[Product behavior](#section_135d0ebe5c4c4a9499bf1811eccb9f4a) 193

Protocol Details

[overview](#section_52b5da79b22248fdb408dd4c67464f9a) 114

Q

[Query Notifications header](#section_e168d373a7b741aab6ca25985466a7e0) 34

R

[References](#section_3ce48492a6d14fbd9f06daf08b48b3cd) 10

[informative](#section_5e29795df0ab4f25b1f5f65bea469f28) 11

[normative](#section_559375dcef824a2ca36ec5ec40c83f9d) 10

[Relationship to other protocols](#section_b33d45ec5c8b4441bf60dce9880bd270) 14

[Remote procedure call](#section_26327437aa3c4e969bba73a6e862ba21) 19

[RPC client request example](#section_1469b2fa6cab42e991f9044d358b306b) 155

[RPC server response example](#section_a4e706cf8ae7480987a1ce293a6f7f68) 157

S

Security

[implementer considerations](#section_5d627db5467d40249aa8da067f419096) 192

[parameter index](#section_30acd29386b242338307d02ac3b7d55c) 192

[Sent Attention state](#section_61fb4f9d95ec41718d4bdac4eb16149f) 126

[Sent Client Request state](#section_c89701263cbc446f9275cc36ba62a2f4) 126

[Sent Initial PRELOGIN Packet state](#section_cc823ca848674387819dcc5c19da5732) 123

[Sent LOGIN7 Record with SPNEGO Packet state](#section_c373a90b113d4c5f826ad410365ee883) 125

[Sent LOGIN7 Record with Standard Login state](#section_ab7a745416a949ef8f208a0de2b31fbb) 125

[Sent TLS/SSL Negotiation Packet state](#section_d62e225bd8654ccc8f73de1ef49e30d4) 124

Sequencing rules

client ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.2.5](#section_bd3a16dd75b64546933059c3ef44d50e) 123)

[Final state](#section_07a0c71e1a7b44558da47e43a09589bc) 127

[Logged In state](#section_e2e7293f55cf43eeb0282d6dea625db6) 126

[Sent Attention state](#section_61fb4f9d95ec41718d4bdac4eb16149f) 126

[Sent Client Request state](#section_c89701263cbc446f9275cc36ba62a2f4) 126

[Sent Initial PRELOGIN Packet state](#section_cc823ca848674387819dcc5c19da5732) 123

[Sent LOGIN7 Record with SPNEGO Packet state](#section_c373a90b113d4c5f826ad410365ee883) 125

[Sent LOGIN7 Record with Standard Login state](#section_ab7a745416a949ef8f208a0de2b31fbb) 125

[Sent TLS/SSL Negotiation Packet state](#section_d62e225bd8654ccc8f73de1ef49e30d4) 124

server ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.3.5](#section_8d3c50da023f4bc5959c60f38ed22825) 129)

[Client Request Execution state](#section_21c9c608c6bc456fb7eb9f09a90bd282) 133

[Final state](#section_f9157109df6b4f23ab9135a5ca43260d) 134

[Initial state](#section_8bfcbcd21baf47928fff86a63a9e90fa) 129

[Logged In state](#section_7c37e9c36c6544dfbbd81e8fc11a618d) 133

[Login Ready state](#section_1d82a68b57c246e984baed41f7bd0c7c) 130

[SPNEGO Negotiation state](#section_7b617d06e13845f3bcea35619b14ac72) 132

[TLS/SSL Negotiation state](#section_ef1c4791413f4ec7ad475810a514db94) 130

Server

abstract data model ([section 3.1.1](#section_1ffab630e56545efbc63cc4eac4bc1db) 114, [section 3.3.1](#section_1457f2976d7d4b47a4bef35d273b8cae) 128)

higher-layer triggered events ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.3.4](#section_6af826c5a9574438a7c0b3bbd309c69a) 129)

initialization ([section 3.1.3](#section_07dcb45cdb9c4ce981cd02b370cb4809) 114, [section 3.3.3](#section_d53254d96e6f4339aef09439e2269688) 129)

[local events](#section_2ceaed83ecdf479eba465fdcbd888ddc) 119

message processing ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.3.5](#section_8d3c50da023f4bc5959c60f38ed22825) 129)

[Client Request Execution state](#section_21c9c608c6bc456fb7eb9f09a90bd282) 133

[Final state](#section_f9157109df6b4f23ab9135a5ca43260d) 134

[Initial state](#section_8bfcbcd21baf47928fff86a63a9e90fa) 129

[Logged In state](#section_7c37e9c36c6544dfbbd81e8fc11a618d) 133

[Login Ready state](#section_1d82a68b57c246e984baed41f7bd0c7c) 130

[SPNEGO Negotiation state](#section_7b617d06e13845f3bcea35619b14ac72) 132

[TLS/SSL Negotiation state](#section_ef1c4791413f4ec7ad475810a514db94) 130

messages

[attention acknowledgment](#section_9bcc4b79a79141d0bf7673a97e501b35) 21

[error and informational messages](#section_7fffdf0c78d140fd9dee1e6b4246d3aa) 21

[login response](#section_da5a73bbe49247ab9d9c23ac00292efc) 20

[overview](#section_342f4cbb2b4b489c8b63f99b12021a94) 19

[pre-login response](#section_63c37a5d459d40b98c4a47e13aff235d) 20

[response completion](#section_6ee2125f34be43c6a38ce9ee7338500f) 21

[return parameters](#section_43acbd448d754f2e8f5c543b59a9d517) 20

[return status](#section_ed493f4cf63d41739476e64261400119) 20

[row data](#section_cc5aaa57f2e04571910ac5f6a538f88f) 20

[other local events](#section_388d89cc00a34f57bc22624e251e68f3) 134

overview ([section 3.1](#section_34e6da7dd80c4875b1553be341822623) 114, [section 3.3](#section_7deaba0f836a4f10a876580dbd78bf49) 127)

sequencing rules ([section 3.1.5](#section_5a9d49b8d1c44ff58f6576dc924f5c4d) 114, [section 3.3.5](#section_8d3c50da023f4bc5959c60f38ed22825) 129)

[Client Request Execution state](#section_21c9c608c6bc456fb7eb9f09a90bd282) 133

[Final state](#section_f9157109df6b4f23ab9135a5ca43260d) 134

[Initial state](#section_8bfcbcd21baf47928fff86a63a9e90fa) 129

[Logged In state](#section_7c37e9c36c6544dfbbd81e8fc11a618d) 133

[Login Ready state](#section_1d82a68b57c246e984baed41f7bd0c7c) 130

[SPNEGO Negotiation state](#section_7b617d06e13845f3bcea35619b14ac72) 132

[TLS/SSL Negotiation state](#section_ef1c4791413f4ec7ad475810a514db94) 130

timer events ([section 3.1.6](#section_9d65081e87bd4461a6e4ff4bb4895b3c) 119, [section 3.3.6](#section_c0d2d1a29c3d4eb68a460a7dab3b3f56) 134)

timers ([section 3.1.2](#section_2192569232bd484d974bd31e025becc8) 114, [section 3.3.2](#section_25a86907cd7a44a39c923b9a9c0e127f) 129)

[Server Messages message](#section_342f4cbb2b4b489c8b63f99b12021a94) 19

[SparseColumn select statement example](#section_b08d4f24b4bd4b5ebda7d74071ae56a1) 163

[SPNEGO Negotiation state](#section_7b617d06e13845f3bcea35619b14ac72) 132

[SQL batch client request example](#section_b05b006b3cbf404bbcaf7ec584b54706) 152

[SQL batch server response example](#section_d2a06235fbef4223b630a90dea24a3d7) 153

[SQL command](#section_5b7f9062d5144b4ea7c59a27a19596c4) 18

[SQL command with binary data](#section_88176081df754b24bcfb4c16ff03cbfa) 18

[SQL command with binary data example](#section_43dac55aa31a4122ac0620cfedf76811) 159

[SSPI message example](#section_dc57840ad13b43a1aad35425afadbc0c) 158

[Standards assignments](#section_9d8480d77c334592a66f48d8bf927662) 16

Syntax

client messages

[Attention](#section_dc28579f49b14a789c5f63fbda002d2e) 19

[login](#section_f159ec89ef1a4fcd9ffd8ee701fccbb0) 18

[overview](#section_7ea9ee1ab46141f29004141c0e712935) 17

[pre-login](#section_58886b79ec7542f3bd7be32f327366b6) 18

[remote procedure call](#section_26327437aa3c4e969bba73a6e862ba21) 19

[SQL command](#section_5b7f9062d5144b4ea7c59a27a19596c4) 18

[SQL command with binary data](#section_88176081df754b24bcfb4c16ff03cbfa) 18

[transaction manager request](#section_b4b7856454404fc0b5efc9e1925aaefe) 19

grammar definition for token description

[data packet stream tokens](#section_f79bb5b85919439aa69648064b78b091) 50

[data stream types](#section_2a626d8c4bba4f58a2c60328aa1ba8cb) 31

[data type definitions](#section_ffb02215af074b5085451fd522106c68) 35

[general rules](#section_d2ed21d6527b46ac803594f6f68eb9a8) 27

[overview](#section_b7b3f6ba65dc491eb84ddbb12654228d) 27

[packet data stream headers](#section_e17e54ae0fac48b7b8a8c267be297923) 33

[TYPE\_INFO rule definition](#section_cbe9c510eae64b1f9893a098944d430a) 48

[overview](#section_a734db771cb247e4b61a0c9dbd8f960f) 17

packet data token and tokenless data streams

[DONE and attention tokens](#section_ef2ba0a16e4b46d09aceb6cc849e417a) 27

[overview](#section_dc3a08548230482fbbb9d94a3b905a26) 25

[token stream](#section_7cfc401bf9b6404da44722e21e593742) 26

[tokenless stream](#section_91e16d3104c7485fa194a649419cad31) 25

packet data token stream definition

[ALTMETADATA](#section_004bba4a8c234d7bab2cd9e7ba864cd0) 78

[ALTROW](#section_d1c427616a6443ab8a55fccb210ac073) 80

[COLINFO](#section_aa8466c5ca3d48caa6387c1becebe754) 81

[COLMETADATA](#section_58880b9f381c43b2bf8b0727a98c4f4c) 82

[DONE](#section_3c06f11098bd4d5bb836b1ba66452cb7) 85

[DONEINPROC](#section_43e891c5f7a1432f8f9f233c4cd96afb) 87

[DONEPROC](#section_65e24140edea46e5b710209af2016195) 88

[ENVCHANGE](#section_2b3eb7e5d43d4d1bbf4d76b9e3afc791) 89

[ERROR](#section_9805e9fa1f8b4cf88f788d2602228635) 93

[FEATUREEXTACK](#section_2eb82f8e11f046dcb42d27302fa4701a) 95

[INFO](#section_284bb815d0834ed5b33abdc2492e322b) 100

[LOGINACK](#section_490e563dcc6e4c86bb95ef0186b98032) 101

[NBCROW](#section_c0da3cc94441476baa995a518f22dc0a) 102

[OFFSET](#section_8d0b37ff20c1439e8f311d7f136249b5) 104

[ORDER](#section_252759be9d744435809dd55dd860ea78) 104

[overview](#section_67b6113cd72242d1902c3f6e8de09173) 77

[RETURNSTATUS](#section_c719f199e71b418790b994f78bd1870e) 105

[RETURNVALUE](#section_7091f6f6b83d4ed2afebba5013dfb18f) 106

[ROW](#section_3840ef933b104aca9fd1a210b8bb6d0c) 108

[SESSIONSTATE](#section_626fbe19f3564599ba17c70f44005106) 109

[SSPI](#section_07e2bb7b8ba6445f89b1cc76d8bfa9c6) 111

[Table Valued Parameter row](#section_103eb1d15514418296ece95896904f50) 112

[TABNAME](#section_140e3348da08409ab6c3f0fc9cee2d6e) 111

packet header message type - stream definition

[BulkLoad - UpdateText/WriteText](#section_6ee5bed798d844f992177b10e2123596) 51

[BulkLoadBCP](#section_ab4a7d62cd1f4db1b67decae58f493e3) 50

[LOGIN7](#section_773a62b6ee894c029e5e344882630aac) 53

[PRELOGIN](#section_60f5640801884cd58b9025c6f2423868) 65

[RPCRequest](#section_619c43b694954a589e49a4950db245b3) 69

[SQLBatch](#section_f2026cd39a464a3f9a08f63140bcbbe3) 73

[SSPI message](#section_1dc6c197f606410badd9e6fe1dff0e8b) 73

[transaction manager request](#section_0fb28ba5ddcb4d0295c3aa5b05ec6092) 74

packets

[overview](#section_e5ea85201ea34a75a2a9c17e63e9ee19) 21

[packet data](#section_3420734b287b4cdea0dd538c770a89a1) 24

[packet header](#section_7af536671b72470382587984e838f746) 22

server messages

[attention acknowledgment](#section_9bcc4b79a79141d0bf7673a97e501b35) 21

[error and informational messages](#section_7fffdf0c78d140fd9dee1e6b4246d3aa) 21

[login response](#section_da5a73bbe49247ab9d9c23ac00292efc) 20

[overview](#section_342f4cbb2b4b489c8b63f99b12021a94) 19

[pre-login response](#section_63c37a5d459d40b98c4a47e13aff235d) 20

[response completion](#section_6ee2125f34be43c6a38ce9ee7338500f) 21

[return parameters](#section_43acbd448d754f2e8f5c543b59a9d517) 20

[return status](#section_ed493f4cf63d41739476e64261400119) 20

[row data](#section_cc5aaa57f2e04571910ac5f6a538f88f) 20

T

[Table response with SESSIONSTATE token data example](#section_f3a3d7443b7c4b7792fdd61bca047ef0) 178

Timer events

client ([section 3.1.6](#section_9d65081e87bd4461a6e4ff4bb4895b3c) 119, [section 3.2.6](#section_030af85119fd4a9ca7df05b637f7d888) 127)

server ([section 3.1.6](#section_9d65081e87bd4461a6e4ff4bb4895b3c) 119, [section 3.3.6](#section_c0d2d1a29c3d4eb68a460a7dab3b3f56) 134)

Timers

client ([section 3.1.2](#section_2192569232bd484d974bd31e025becc8) 114, [section 3.2.2](#section_8313b90eeee347e1844173b3b5df1e39) 121)

server ([section 3.1.2](#section_2192569232bd484d974bd31e025becc8) 114, [section 3.3.2](#section_25a86907cd7a44a39c923b9a9c0e127f) 129)

[TLS/SSL Negotiation state](#section_ef1c4791413f4ec7ad475810a514db94) 130

Token data stream

[overview](#section_7cfc401bf9b6404da44722e21e593742) 26

token definition

[fixed-length token](#section_5d73136323384ebeb1eb5cce0780ef8f) 26

[overview](#section_427e90dfa7284899aafff42a0c9bbd1a) 26

[variable-count tokens](#section_5ff30ab3aaa7474ba095662fcaed5549) 27

[variable-length tokens](#section_d3edea23be1f416098f50de233ffeebc) 27

[zero-length token](#section_9b571ae55d0048748cf798f30ca69dbd) 26

Token data stream definition

[ALTMETADATA](#section_004bba4a8c234d7bab2cd9e7ba864cd0) 78

[ALTROW](#section_d1c427616a6443ab8a55fccb210ac073) 80

[COLINFO](#section_aa8466c5ca3d48caa6387c1becebe754) 81

[COLMETADATA](#section_58880b9f381c43b2bf8b0727a98c4f4c) 82

[DONE](#section_3c06f11098bd4d5bb836b1ba66452cb7) 85

[DONEINPROC](#section_43e891c5f7a1432f8f9f233c4cd96afb) 87

[DONEPROC](#section_65e24140edea46e5b710209af2016195) 88

[ENVCHANGE](#section_2b3eb7e5d43d4d1bbf4d76b9e3afc791) 89

[ERROR](#section_9805e9fa1f8b4cf88f788d2602228635) 93

[FEATUREEXTACK](#section_2eb82f8e11f046dcb42d27302fa4701a) 95

[INFO](#section_284bb815d0834ed5b33abdc2492e322b) 100

[LOGINACK](#section_490e563dcc6e4c86bb95ef0186b98032) 101

[NBCROW](#section_c0da3cc94441476baa995a518f22dc0a) 102

[OFFSET](#section_8d0b37ff20c1439e8f311d7f136249b5) 104

[ORDER](#section_252759be9d744435809dd55dd860ea78) 104

[overview](#section_67b6113cd72242d1902c3f6e8de09173) 77

[RETURNSTATUS](#section_c719f199e71b418790b994f78bd1870e) 105

[RETURNVALUE](#section_7091f6f6b83d4ed2afebba5013dfb18f) 106

[ROW](#section_3840ef933b104aca9fd1a210b8bb6d0c) 108

[SESSIONSTATE](#section_626fbe19f3564599ba17c70f44005106) 109

[SSPI](#section_07e2bb7b8ba6445f89b1cc76d8bfa9c6) 111

[Table Valued Parameter row](#section_103eb1d15514418296ece95896904f50) 112

[TABNAME](#section_140e3348da08409ab6c3f0fc9cee2d6e) 111

Token data stream examples

[out-of-band attention signal](#section_f8aa004ebf764f94bd5b39597a888146) 180

[overview](#section_ee3cc0fa23474a10b22f9e316470e826) 180

[sending an SQL batch](#section_0d27294c86d5400d897faeedca6c4e13) 180

Token description - grammar definition

[data packet stream tokens](#section_f79bb5b85919439aa69648064b78b091) 50

data stream types

[data type dependent data streams](#section_3f983fde0509485a8c40a9fa6679a828) 31

[unknown-length data streams](#section_1893c1ffedce4465b194022039c98928) 31

[variable-length data streams](#section_47acf60e9c364184b016a4947263e25f) 31

data type definitions

[fixed-length data types](#section_859eb3d280d340f6a637414552c9c552) 35

[overview](#section_ffb02215af074b5085451fd522106c68) 35

[partially length-prefixed data types](#section_7d26a257083e409b81ba897e0c672be0) 39

[SQL\_VARIANT](#section_2435e85d9e61492cacb2627ffccb5b92) 42

[Table Valued Parameter](#section_c264db71c1ec4fe8b5ef19d54b1e6566) 43

[UDT Assembly Information](#section_a57df60ed0a64e7ea2e5ccacd277c673) 41

[variable-length data types](#section_ce3183a69d8947e8a02fde5a1a1303de) 36

[XML data type](#section_bfebeca7a3a643dea7e7fa721e803b4e) 42

general rules

[collation rule definition](#section_3d29e8dc218a42c69ba4947ebca9fd7e) 30

[least significant bit order](#section_bbc22f15e1a04338a169c79819d39b1c) 30

[overview](#section_d2ed21d6527b46ac803594f6f68eb9a8) 27

[overview](#section_b7b3f6ba65dc491eb84ddbb12654228d) 27

packet data stream headers

[overview](#section_e17e54ae0fac48b7b8a8c267be297923) 33

[Query Notifications header](#section_e168d373a7b741aab6ca25985466a7e0) 34

[Transaction Descriptor header](#section_4257dd95ef6c4621b75d270738487d68) 34

[TYPE\_INFO rule definition](#section_cbe9c510eae64b1f9893a098944d430a) 48

[Tokenless data stream](#section_91e16d3104c7485fa194a649419cad31) 25

[Tracking changes](#section_47286277defd46f3a074e769bf9dab1e) 199

[Transaction Descriptor header](#section_4257dd95ef6c4621b75d270738487d68) 34

[Transaction manager request](#section_b4b7856454404fc0b5efc9e1925aaefe) 19

[Transaction manager request example](#section_9f10990a9c744a3989da2a3b05effacb) 160

[Transport](#section_fd30432f71b2488cb30f19737d76d970) 17

Triggered events - higher-layer

client ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.2.4](#section_e7af6058672941438815f82a475f9ba6) 122)

server ([section 3.1.4](#section_011f443e690b4f37a06999fe7589fe0c) 114, [section 3.3.4](#section_6af826c5a9574438a7c0b3bbd309c69a) 129)

[TVP insert statement example](#section_2821fec5e1f642d281e840a2b7fead2d) 161

U

[Unknown-length data streams](#section_1893c1ffedce4465b194022039c98928) 31

V

[Variable-count tokens](#section_5ff30ab3aaa7474ba095662fcaed5549) 27

[Variable-length data streams](#section_47acf60e9c364184b016a4947263e25f) 31

[Variable-length tokens](#section_d3edea23be1f416098f50de233ffeebc) 27

[Vendor-extensible fields](#section_f3a6041cfe84420abda941021dee1cbe) 16

[Versioning](#section_6809de85166544ada4596f6621e02257) 15

Z

[Zero-length token](#section_9b571ae55d0048748cf798f30ca69dbd) 26