Connection of a SIMATIC S7-1x00 to a SQL Database

SQL / Tabular Data Stream (SQL)

https://support.industry.siemens.com/cs/ww/en/view/109779336

Siemens Industry Online Support

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: https://www.siemens.com/industrialsecurity.

Table of Contents

1 Introduction

1.1 Overview

Scenario

This application example lets the user write a continuous stream of data directly from the user program of a controller to a database or pull data from a database without having to implement an additional layer.

Tabular Data Stream (TDS) is a TCP-level protocol that enables data exchange between a Microsoft SQL Server and a client. TDS allows the controller to implement access to the database via Open User Communication (OUC).

With TDS you can log into a SQL server database and transfer SQL commands. This allows data to be read from the database or sent there for storage.

This application example demonstrates how a SIMATIC S7-1500 based on the Open User Communication blocks (TCON, TSEND, TRCV and TDISCON) establishes a TCP connection to a Microsoft SQL Server and exchanges data with a database.

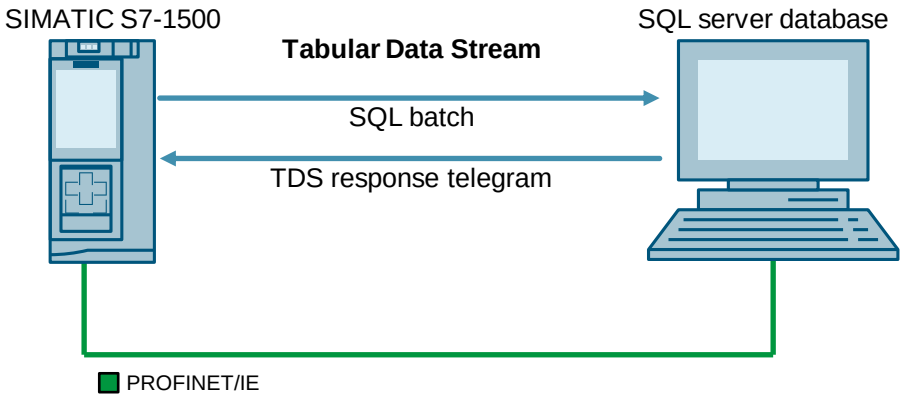
The following accesses to a Microsoft SQL Server database are implemented in the application example.

- PRELOGIN: Send message in order to set up the context for the login.
- LOGIN: Log in to a Microsoft SQL server database.
- Formulate and send SQL batch to transmit SQL commands:
  - SELECT
  - INSERT INTO
  - UPDATE
- Formulate and send SQL batch to execute stored procedures.
- Response received from SQL server to the executed SQL command.

Overview of the application example

The Figure below provides an overview of the application example.

Figure 1-1



1.2 Operating Principle

Overview

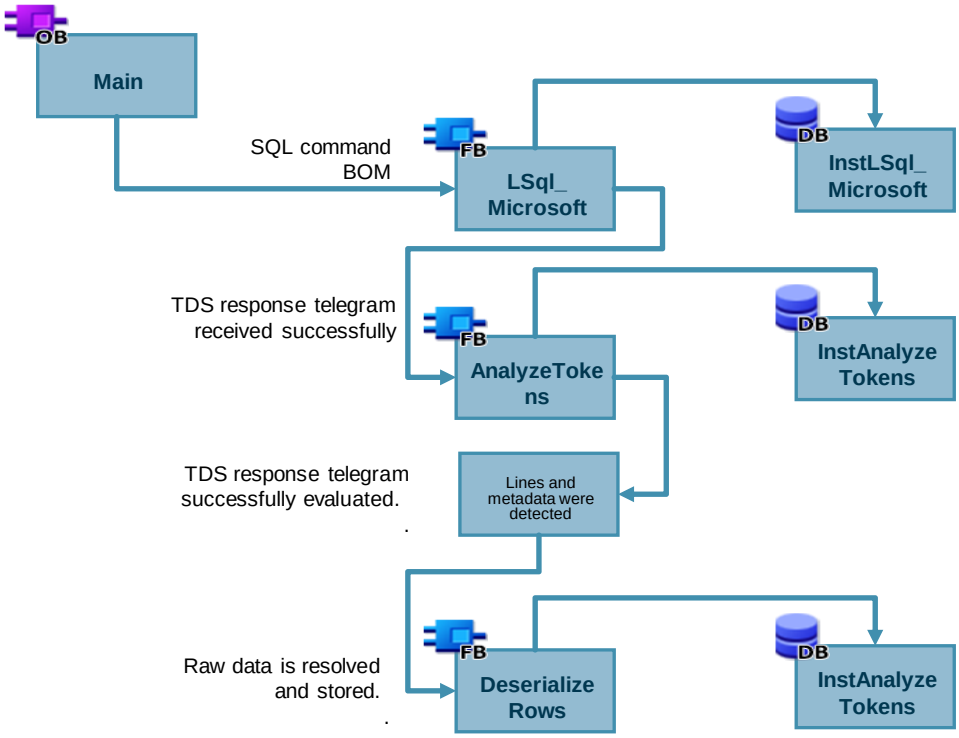
This application example contains the following function blocks (FBs):

Table 1-1

Function block (FB)	Description
FB "LSql_Microsoft"	The FB "LSql_Microsoft" executes the accesses to the Microsoft SQL Server database.
FB "AnalyzeTokens"	The FB "AnalyzeTokens" evaluates the response of the SQL server to the executed SQL command. The Tabular Data Stream (TDS) is divided into TDS header, packet data, and DONE token. **TDS header** The content of the TDS header is output at the "tdsHeader" output. **Packet data** The messages contained in the packet data themselves contain the following token streams: • ColumnMetaData: Metadata of the SQL columns Detailed information about the token stream "ColumnMetaData" can be found at the following link: https://www.freetds.org/tds.html#types The following data of the token stream "ColumnMetaData" are output at the outputs of the FB "AnalyzeTokens": - Number of columns - Properties of the SQL columns • Row data: Data from the read rows The following data of the token stream "Row data" are output at the outputs of the FB "AnalyzeTokens": - Address of the first byte of the "Row data" token stream - Length of the token stream "Row data" • Error: SQL server error message and information. **DONE token** The DONE token indicates the end of the TDS response telegram. It contains the number of rows that were processed. The number of processed lines is output at the output of the FB "AnalyzeTokens".
FB "DeserializeRows"	The FB "DeserializeRows" resolves the raw data of the rows transferred in the token stream "Row data" and stores it according to the data type used in an use-case specific data structure.

The following Figure shows the call hierarchy of the FBs in OB1:

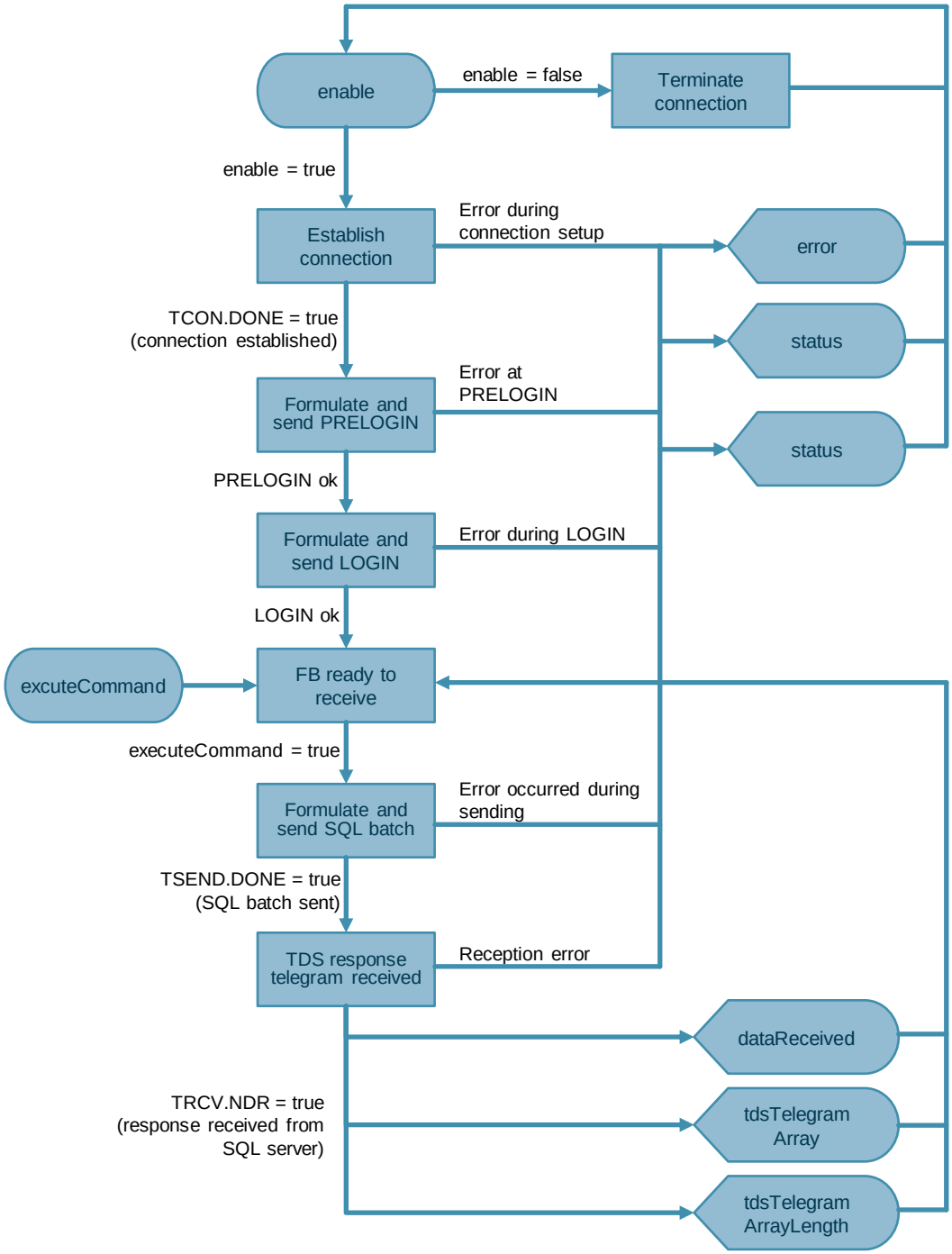
Figure 1-2



Function block (FB) "LSql_Microsoft"

The Figure below shows the operating principle and the structure of the "LSql_Microsoft" FB.

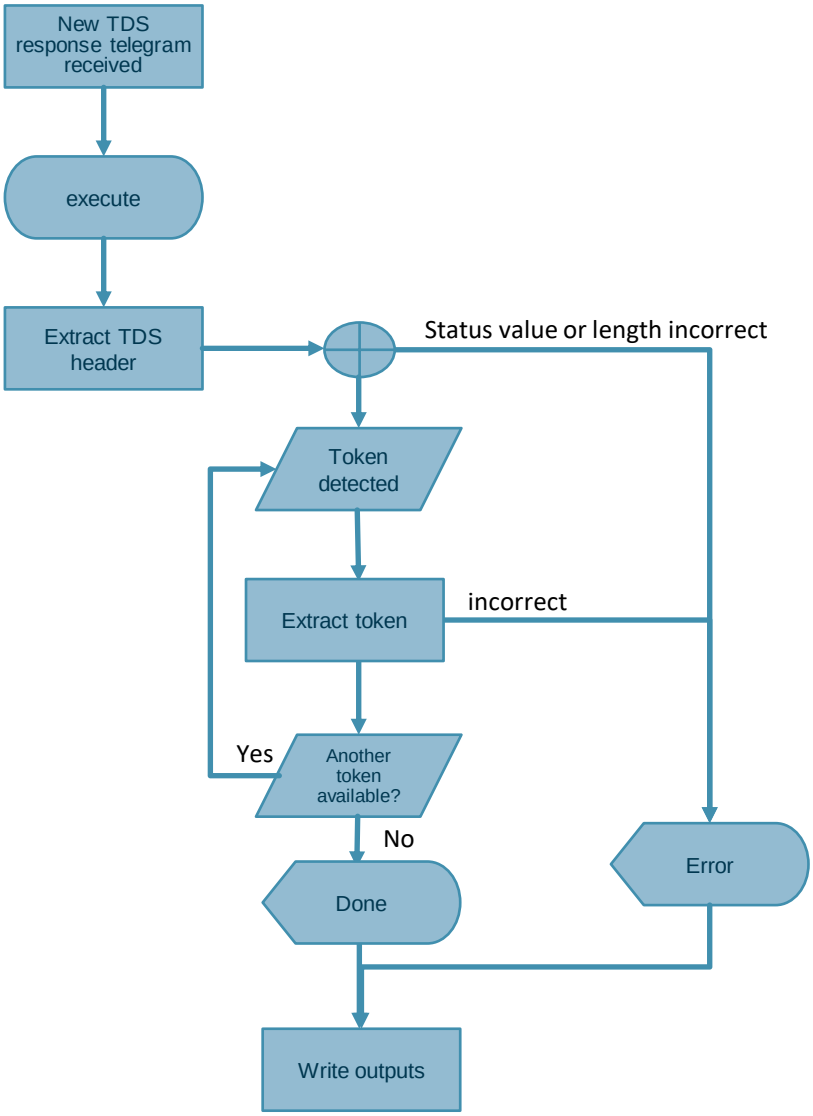
Figure 1-3



"AnalyzeTokens" function block (FB)

The following Figure shows the operating principle and structure of the FB "AnalyzeTokens".

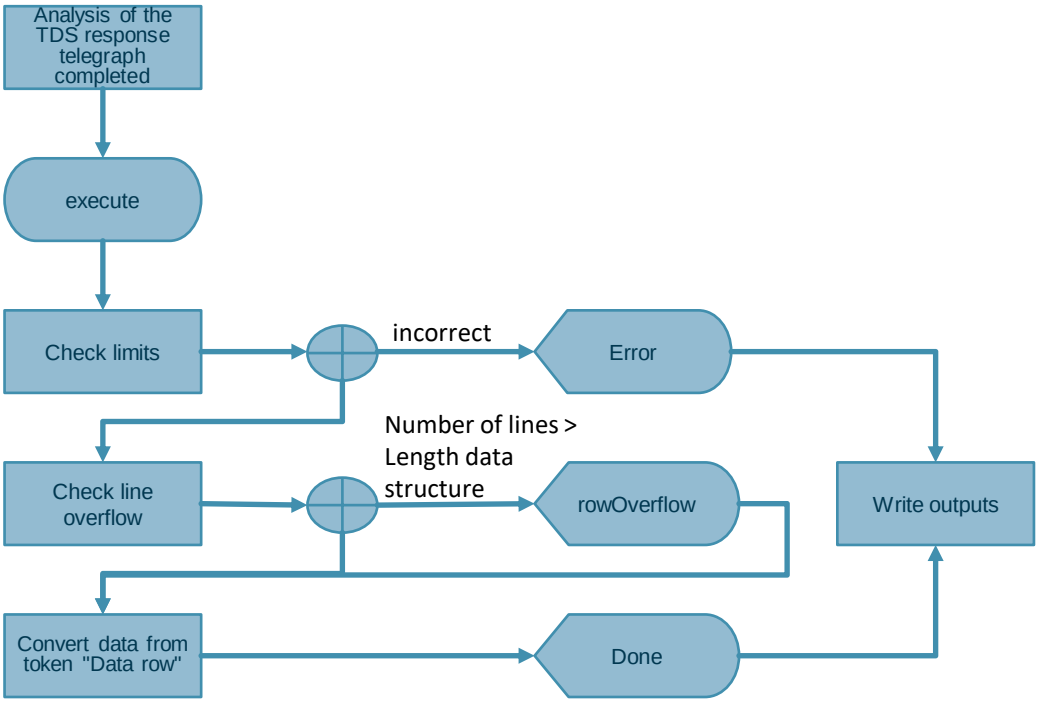
Figure 1-4



Function block (FB) "DeserializeRows"

The Figure below shows the operating principle and the structure of the "DeserializeRows" FB.

Figure 1-5



1.3 Components Used

This application example was created with these hardware and software components:

Table 1-2

Components	Quantity	Item Number	Note
STEP 7 Professional V17 Update 2	1	6ES7822-1AA07-0YA7	Engineering System
CPU 1517F-3 PN	1	6ES7517-3FP00-0AB0	Alternatively, you can use another S7-1500 CPU or ET 200SP CPU as of firmware V2.5.

The listed components can be obtained from the Siemens Industry Mall, for example.

This application example consists of the following components:

Table 1-3

Components	File Name	Note
Documentation	109779336_SQL_DOC_de_V31.pdf	This document
Project	109779336_SQL_CODE_V31.zip	This zipped file contains the STEP 7 project of the application example for S7-1500 CPUs and S7-1200 CPUs.
Library	109779336_SQL_LIB_V31.zip	This zipped file contains the library of the application example for S7-1500 CPUs and S7-1200 CPUs.

2 Engineering

2.1 Interface Description

2.1.1 FB "LSql_Microsoft"

Functional description

The "LSql_Microsoft" FB emulates the TDS protocol on the basis of "Open User Communication blocks". It facilitates the following actions:

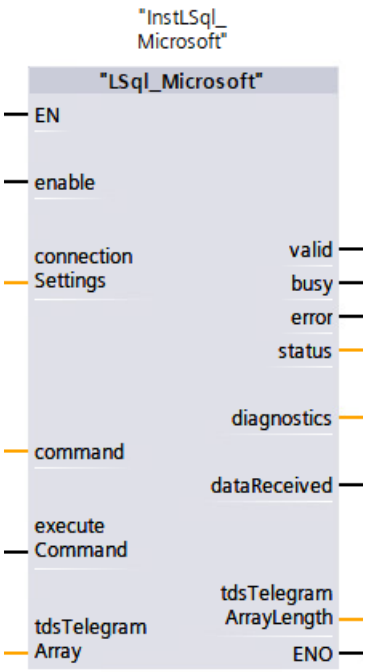
- Connecting and logging into a Microsoft SQL Server database (enable, connectionSettings)
- Transfer SQL command (sqlCommand, executeCommand)
- Receive read data (tdsTelegrammArray, tdsTelegrammmArrayLength, dataReceived)

Internally, the block works with additional self-created functions (FCs). They are not explained here in more detail.

Block interface

The following Figure shows the interfaces of the function block "LSql_Microsoft" and the associated data types.

Figure 2-1



The Table below shows the inputs and outputs of the "LSql_Microsoft" FB.

Table 2-1

Name	P Type	Data Type	Description
enable	IN	Bool	Activates the function of the FB.
connectionSettings	IN	"LSql_typeConnectionSettings"	Parameters for establishing a connection and logging into the database. Detailed information about the PLC data type "LSql_typeConnectionSettings" can be found in Table 2-4.
command	IN	String	SQL command that will be executed if executeCommand = TRUE.
executeCommand	IN	Bool	TRUE: SQL command is executed once.
valid	OUT	Bool	TRUE: valid values available at the outputs of the FB.
busy	OUT	Bool	TRUE: FB is not ready yet and new values at the outputs are expected.
error	OUT	Bool	TRUE: An error occurred during the execution of the FB.
status	OUT	Word	• 16#0000 - 16#7FFF: Status of the FB • 16#8000 - 16#FFFF: Error detection Detailed information on the status and error messages can be found in the Section 2.5.
diagnostics	OUT	"LSql_typeDiagnostics"	Diagnostic information of the FB. Detailed information about the PLC data type "LSql_typeDiagnostics" can be found in Table 2-11.
dataReceived	OUT	Bool	TRUE: New data is available at the outputs.
tdsTelegramArrayLength	OUT	UDInt	Length (number of bytes) of the received TDS response telegram.
tdsTelegramArray	IN_OUT	Array[*] of bytes	Receive buffer for the TDS response telegram.

2.1.2 FB "AnalyzeTokens"

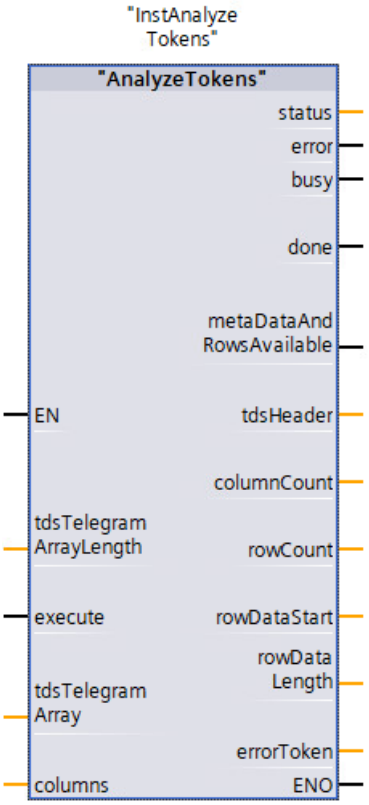
Functional description

The FB "AnalyzeTokens" evaluates the received TDS response telegram.

Block interface

The following Figure shows the interfaces of the "AnalyzeTokens" function block and the associated data types.

Figure 2-2



The following table shows the inputs and outputs of the FB "AnalyzeTokens".

Table 2-2

Name	P Type	Data Type	Description
tdsTelegramArrayLength	IN	UDInt	Number of bytes received via TDS.
execute	IN	Bool	TRUE: Execution of the FB is initiated.
status	OUT	Word	Status display
error	OUT	Bool	Error display
busy	OUT	Bool	TRUE: FB is in preparation.
done	OUT	Bool	TRUE: The FB is done working and new values are available at the outputs of the FB.
metaDataAndRowsAvailable	OUT	Bool	Lines and metadata are available.

Name	P Type	Data Type	Description
tdsHeader	OUT	"LSql_typePacketHeader"	TDS header Detailed information about the PLC data type "LSql_typePacketHeader" can be found in Table 2-7.
columnCount	OUT	UInt	Number of columns received.
rowCount	OUT	UDInt	Number of lines received.
rowDataStart	OUT	UInt	Address of the first byte of the array "tdsTelegramArray".
rowDataLength	OUT	UInt	Length of the token stream "Row data".
errorToken	OUT	"Lsql_typeErrorToken"	Structure for the ERROR token of the TDS response telegram Detailed information about the PLC data type "LSql_typeErrorToken" can be found in Table 2-12.
tdsTelegramArray	IN_OUT	Array[*] of bytes	Receive buffer in which the received bytes of the TDS response telegram are stored.
columns	IN_OUT	Array[*] of "LSql_typeColumn"	Properties of the columns. Detailed information about the PLC data type "LSql_typeColumn" can be found in Table 2-8.

2.1.3 FB "DeserializeRows"

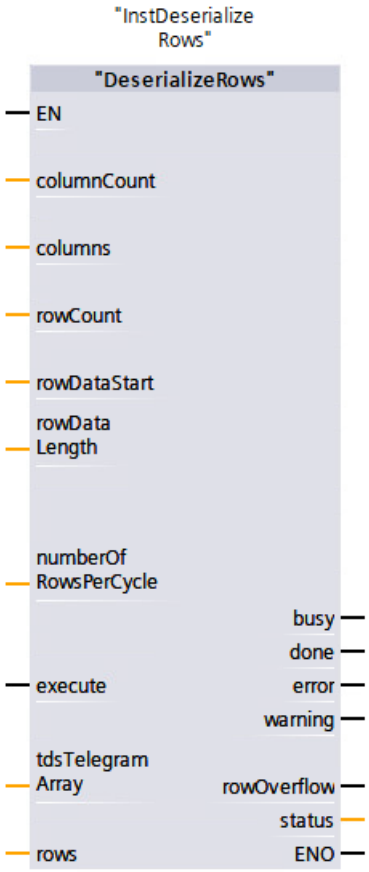
Functional description

The FB "DeserializeRows" resolves the raw data of the read rows and stores the values according to the used data type in an application case specific data structure.

Block interface

The following Figure shows the interfaces of the function block "DeserializeRows" and the associated data types.

Figure 2-3



The Table below shows the inputs and outputs of the "DeserializeRows" FB.

Table 2-3

Name	P Type	Data Type	Description
columnCount	IN	UInt	Number of columns received.
columns	IN	Array [0.."LSQL_NUMBER_OF_COLUMNS_MAX"] of "LSql_typeColumn"	Properties of the columns. Detailed information about the PLC data type "LSql_typeColumn" can be found in Table 2-8.
rowCount	IN	UDInt	Number of lines received.
rowDataStart	IN	UInt	Address of the first byte of the token stream "Row data".
rowDataLength	IN	UDInt	Length of the token stream "Row data".
numberOfRowsPerCycle	IN	USInt	Number of lines to be converted per cycle.
execute	IN	Bool	TRUE: Execution of the FB is initiated.
busy	OUT	Bool	TRUE: FB is in preparation.
done	OUT	Bool	TRUE: The FB is done working and new values are available at the outputs of the FB.
error	OUT	Bool	Error display
warning	OUT	Bool	Alarm message
status	OUT	Word	Status display
rowOverflow	OUT	Bool	TRUE: More rows were received from TDS than there are elements in the "typeTokenRows" data structure.
tdsTelegramArray	IN_OUT	Array[*] of bytes	Receive buffer in which the received bytes of the TDS response telegram are stored.
rows	IN_OUT	Array[*] of "typeTokenRow"	User-specific data structure where the output rows for the SQL command are stored. The data structure depends on the user data. Detailed information on the structure of the PLC data type can be found in Table 2-9.

2.1.4 PLC Data Types

"LSql_typeConnectionSettings"

Table 2-4

Parameter	Data Type	Description
interfaceSettings	TCON_IP_v4	IPv4 connection parameters
loginInformation	"LSql_typeLoginInformation"	Login data for authentication between client and SQL server. Detailed information on the structure of the PLC data type "LSql_typeLoginInformation" can be found in Table 2-5.

"LSql_typeLoginInformation"

Table 2-5

Parameter	Data Type	Description
hostName	String	Optional: Name of the local host
userName	String	Required: Username for logging in to the database
password	String	Required: Password for logging in to the database
appName	String	Optional: Name of the application connecting with the database.
serverName	String	Required: Server name of the database
libraryName	String	Optional: Name of the user interface
language	String	Optional: Language of the user interface.
databaseName	String	Required: Database being read or written.
sspi	String	Optional/not supported: Encryption via Security Support Provider Interface (SSPI)
attachDbfile	String	Optional: File name to be added during transmission.
changePassword	String	Optional: New password, should the old one be modified.

"typeSqlData"

Table 2-6

Parameter	Data Type	Description
tdsHeader	"LSql_typePacketHeader"	TDS header Detailed information about the structure of the PLC data type "LSql_typePacketHeader" can be found in Table 2-7.
columns	Array[0.."LSQL_NUMBER_OF_COLUMNS_MAX"] of "LSql_typeColumn"	Data structure for the properties of SQL columns. Detailed information about the structure of the PLC data type "LSql_typeColumn" can be found in Table 2-8.
columnCount	UInt	Number of columns.
rowDataStart	UInt	Address of the first byte of the token stream "Row data".
rowDataLength	UDInt	Length of the token stream "Row data".
rowCount	UDInt	Number of lines received.
rowOverflow	Bool	TRUE: More rows were received from TDS than there are elements in the "rows" data structure.
rows	Array[0.. "LSQL_NUMBER_OF_ROWS_MAX"] of "typeTokenRow"	User-specific data structure where the output rows for the SQL command are stored. The data structure depends on the user data. Detailed information on the structure of the PLC data type "typeTokenRow" can be found in Table 2-9.
errorToken	"Lsql_typeErrorToken"	Structure for the error token of the TDS response telegram Detailed information about the PLC data type "LSql_typeErrorToken" can be found in Table 2-12.

"LSql_typePacketHeader"

Table 2-7

Parameter	Data Type	Description
type	Byte	Message type 4 = TDS response telegram from SQL server
status	Byte	Message status 0 = "normal" message 1 = End of message (EOM) The EOM indicates the last packet in the message.
length	UInt	Length of the TDS response telegram.
spId	Word	Process ID on the server corresponding to the current connection.
paketID	Byte	ID of a message for a package.
window	Byte	This parameter is currently unused.

"LSql_typeColumn"

Table 2-8

Parameter	Data Type	Description
userType	UDInt	User type of the column.
columnType	UInt	ID of the column data type.
columnNameLength	Int	Length of the column heading.
columnName	String	Column heading
columnSizeFieldSize	USInt	Size (number of bytes) of the "columnSize" parameter.
columnSize	USInt	Size of the column data.
flags	"LSql_typeColumnFlag"	Flags for the properties of SQL columns. Detailed information on the structure of the PLC data type "LSql_typeColumnFlag" can be found in Table 2-10.

"typeTokenRow"

The following table shows the structure of the user-specific data structure "typeTokenRow" used in this application example. The user-specific data structure "typeTokenRow" stores the output rows for the executed SQL batch.

Table 2-9

Parameter	Data Type	Description
Amount	Int	Value of the row in column "Amount"
Color	String	Value of the row in column "Color"
Fruit	String	Value of the row in the column "Fruit"
Fresh	Bool	Value of the row in the column "Fresh"
Country	String	Value of the row in the column "Origin"
CountryCode	String[5]	Value of the row in the column "CountryCode"
City	String	Value of the row in the column "City"
Key	String[20]	Value of the row in the column "Key"

Note The data structure "typeTokenRow" in this application example is an image of the example database table "PLCDATA_2" (see Section 2.2).

"LSql_typeColumnFlag"

Table 2-10

Parameter	Data Type	Description
nullable	Bool	TRUE: Saving of a null value is not allowed.
caseSensitive	Bool	TRUE: Column search is case-sensitive.
updateable	USInt	0: Column is read-only. 1: Column is readable and writable. 2: Updatability of the column is unknown.
identity	Bool	TRUE: Column is an identity column.
computed	Bool	TRUE: Column is computed.

"LSql_typeDiagnostics"

Table 2-11

Parameter	Data Type	Description
status	Word	Status of the block or error detection when the error occurs.
subfunctionStatus	Word	Status or return value of called FBs, FCs, and system blocks.

"LSql_typeErrorToken"

Table 2-12

Parameter	Data Type	Description
TokenLength	UInt	The total length of the error data in bytes.
SQLErrorNumber	UDInt	SQL error code
State	Byte	Status number of the error
Class	USInt	The class (severity) of the error. A class smaller than 10 corresponds to an information message.
ErrorMessageLength	UInt	The length of the message text in characters.
ErrorMessage	String	Message text.
ServerNameLength	USInt	The length of the server name in characters.
ServerName	String	Server name.
ProcNameLength	USInt	The length of the stored procedure name in characters.
ProcName	String	The stored procedure name.
LineNumber	UDInt	The line number in the SQL batch or stored procedure where the error occurred. The line number starts with 1. If the line number is not relevant for the error message, the value is 0.

2.2 Structure of the Example Database

To run the application example, you need an SQL database.

Note

The installation and setup of Microsoft SQL Server and the database is not part of this application example.
Follow the instructions for setting up and configuring Microsoft SQL Server in Section 3.

The database of this example has the name "SQLFromPLC" and is structured as follows:

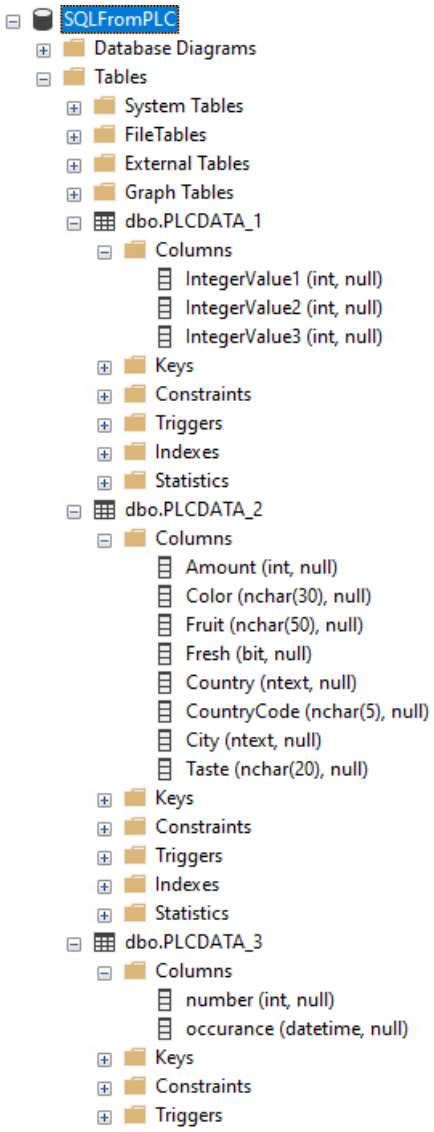
Table 2-13

Table	Column Name	Data Type
PLCDATA_1	IntegerValue1	Int
	IntegerValue2	Int
	IntegerValue3	Int
PLCDATA_2	Amount	Int
	Color	nchar(30)
	Fruit	nchar(50)
	Fresh	bit
	Country	ntext
	CountryCode	nchar(5)
	City	ntext
	Key	nchar(20)
PLCDATA_3	number	Int
	occurrence	datetime

The structure was chosen to demonstrate how to store numbers, strings, and timestamps.

The figure below shows you the database in SQL Server Management Studio.

Figure 2-4



2.3 Integration into the User Project

Requirements

The following requirements apply to the use of the application example:

- S7-1500 as of firmware V2.5
- Microsoft SQL Server is fully configured (see Section 3.2)
- S7-1500 and Microsoft SQL Server are in the same subnet.
- Port "1433" is enabled in the firewall.

Note This block is also functional with a S7-1200 as of firmware V4.4.

Note If the controller and Microsoft SQL Server are located in separate subnets and are connected via a router, the router and a DNS Server address must be configured in the controller and in the SQL server.

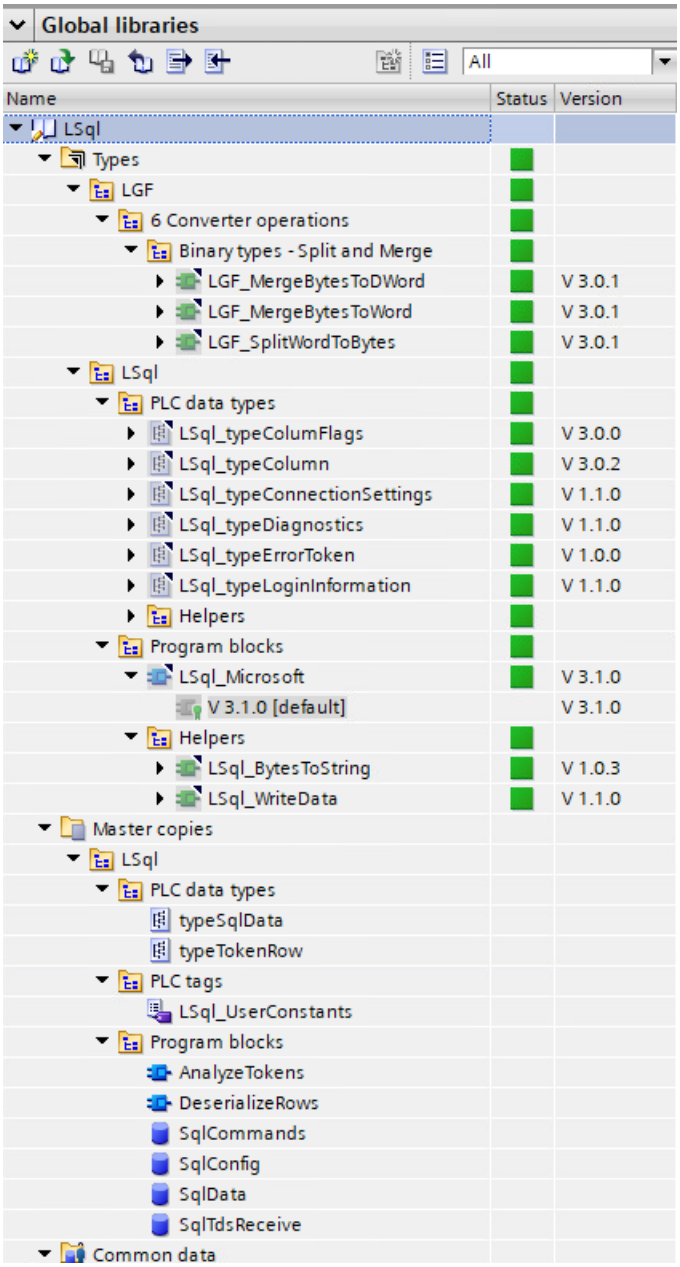
Restrictions

The following restrictions apply for this application example:

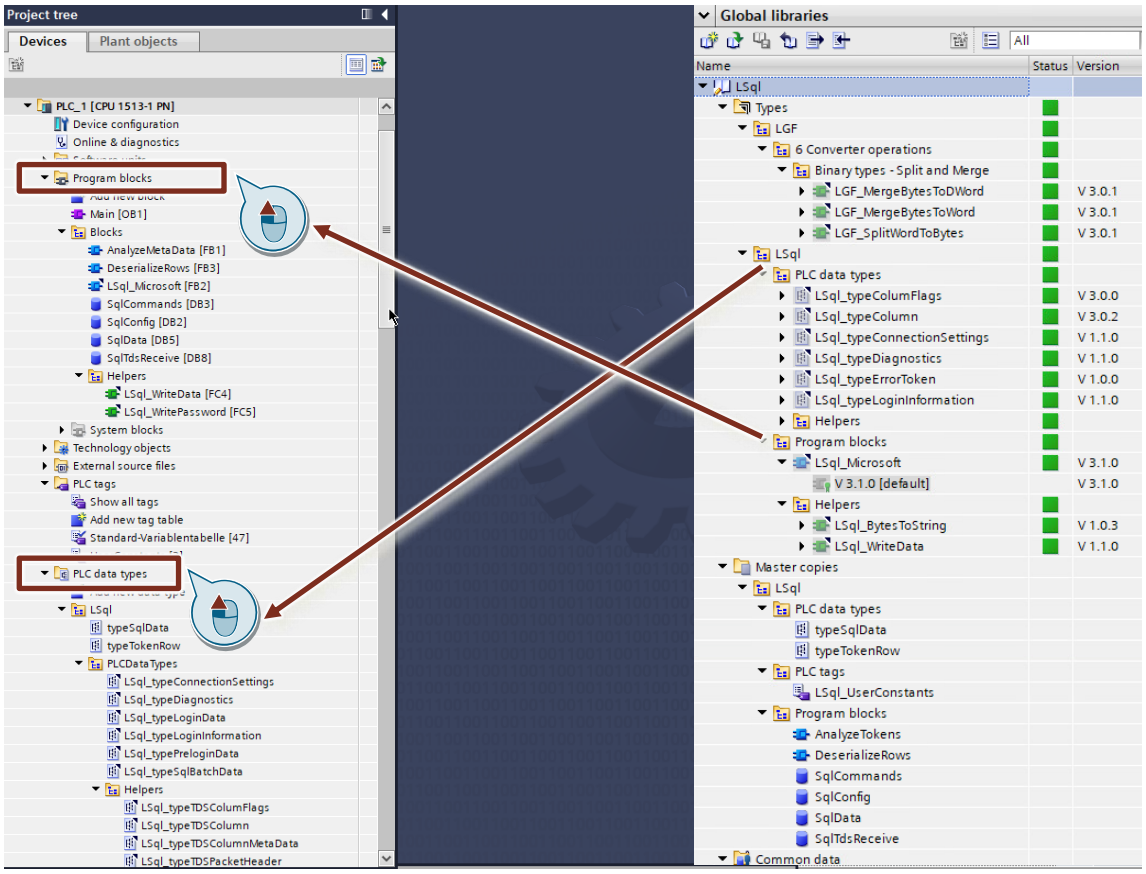
- The application example only works with the tested hardware and software versions.
- Using Open User Communication with an S7-1500, a maximum of 65536 bytes per job can be sent or received.
- Using Open User Communication with an S7-1200, a maximum of 8192 bytes per job can be sent or received.
- The block "LSql_Microsoft" may only be called once per Microsoft SQL Server connection.

Integrate library into user project

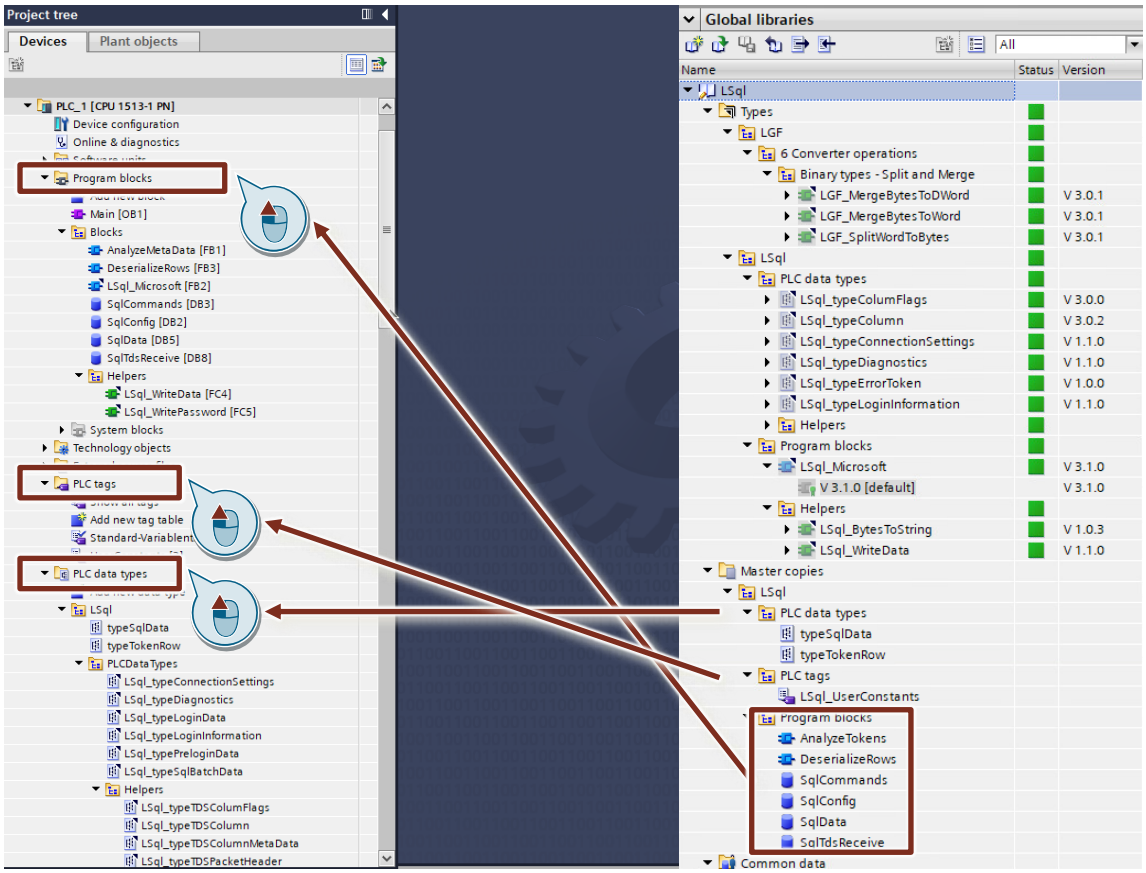
1. Open the "LSql" library in TIA Portal.
   The following Figure shows the "LSql" library.



2. Navigate to the folder "Types > LSql".
3. Copy the following components into your TIA Portal project:
   - Block folder "Blocks"
   - PLC data types folder "PLCDataTypes"



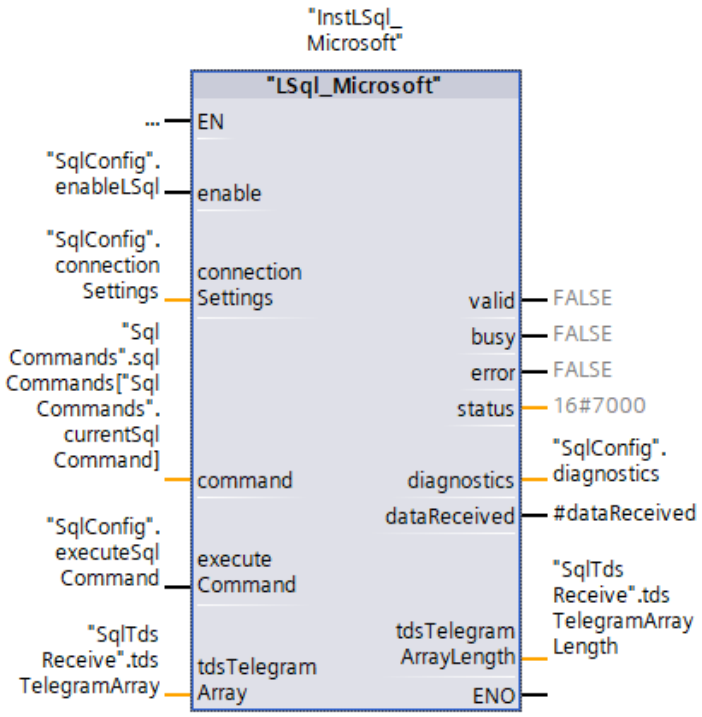
4. Navigate to the folder "Master copies > LSql".
5. Copy the following components into your TIA Portal project:
   - Block folder "Blocks"
   - Data block "SqlConfig"
   - Data block "SqlCommands"
   - Data block "SqlData"
   - Data block "SqlTdsReceive"
   - PLC data types folder "PLCDataTypes"
   - User constants folder "UserConstants"



Connecting the parameters of the FB "LSql_Microsoft"

Call the FB "LSql_Microsoft" in a cyclic block, e.g., "Main [OB1]" and connect the inputs and outputs as seen in the following Figure (minimum connection).

Figure 2-5



Note

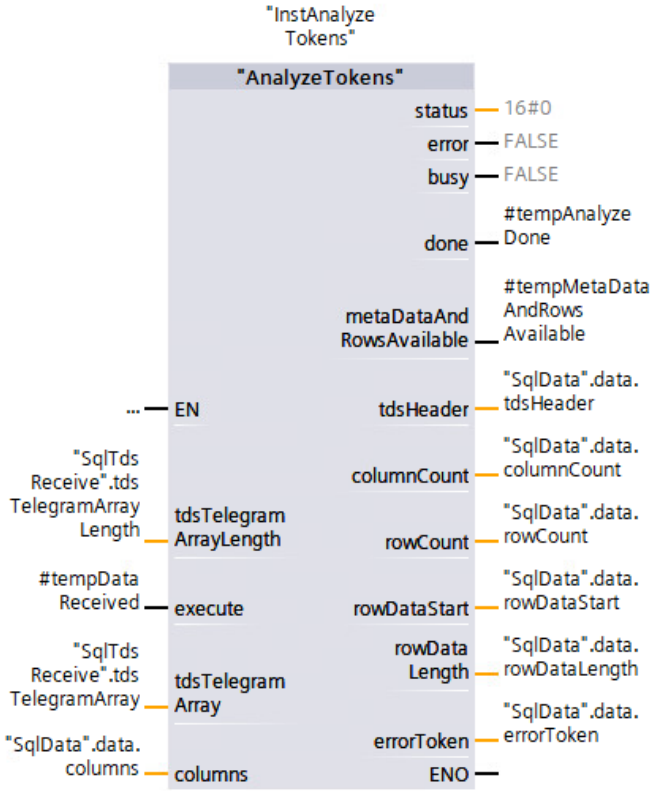
The wiring of the parameters described here is an essential requirement for operating the block.

On the SQL server side, it is necessary to enable the connection settings and credentials to establish the connection. Detailed information on this topic can be found in Section 3.2.

Connection of the parameters of the "AnalyzeTokens" FB

Call the FB "AnalyzeTokens" in a cyclic block, e.g., "Main [OB1]" and connect the inputs and outputs as shown in the following figure (minimum connection).

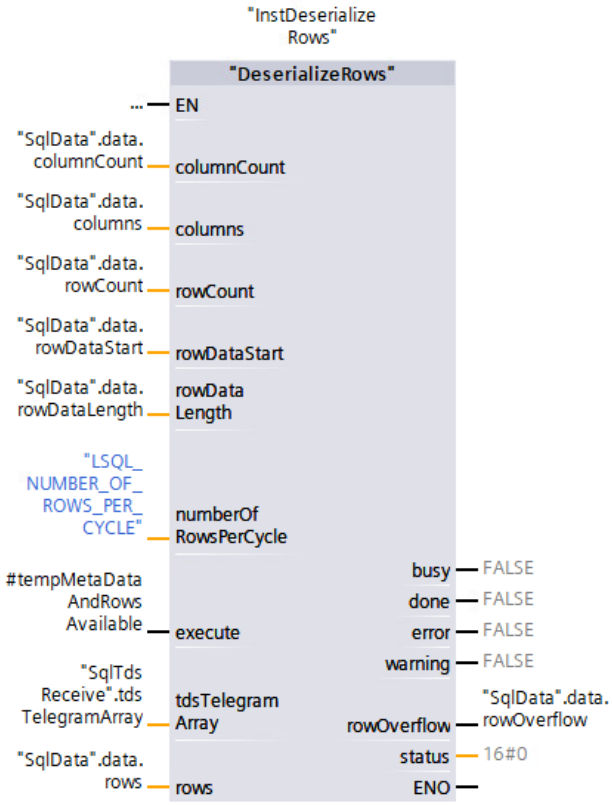
Figure 2-6



Connecting the parameters of the FB "DeserializeRows"

Call the FB "DeserializeRows" in a cyclic block, e.g., "Main [OB1]" and interconnect the inputs and outputs as seen in the following Figure (minimum connection).

Figure 2-7



2.4 Operation of the FB "LSql_Microsoft"

The "LSql_Microsoft" block is controlled via the "enable" and "executeCommand" inputs.

2.4.1 Establishing and terminating the Connection

The "enable" input controls the connection establishment and disconnection to the SQL server. As long as SQL commands are to be transferred to the SQL server, "enable" must be set to "TRUE". If "enable" is set to "FALSE", the connection to the SQL server is broken.

To successfully establish a connection, the following parameters in the "SqlConfig" data block must be set. The unfilled parameters are optional.

Figure 2-8

SqlConfig

Name	Data type	Start value	
Static			
connectionSettings	"LSql_typeConnectionSettings"		
interfaceSettings	TCON_IP_v4		
InterfaceId	HW_ANY	64	
ID	CONN_OUC	16#10	
ConnectionType	Byte	16#0B	
ActiveEstablished	Bool	true	
RemoteAddress	IP_V4		
ADDR	Array[1..4] of Byte		
ADDR[1]	Byte	172	1
ADDR[2]	Byte	16	
ADDR[3]	Byte	43	
ADDR[4]	Byte	198	
RemotePort	UInt	1433	
LocalPort	UInt	0	
loginInformation	"LSql_typeLoginInformation"		
hostName	String	''	
userName	String	'SQL_S71500'	2
password	String	'SQL_S71500'	
appName	String	''	
serverName	String	'SQLEXPRESS'	3
libraryName	String	''	
language	String	''	
databaseName	String	'SQLFromPLC'	4
sspi	String	''	
attachDbfile	String	''	
changePassword	String	''	

Table 2-14

	Parameter	Note
1.	IP address and port of the SQL server	The default port for Microsoft SQL Server is 1433.
2.	SQL server credentials	See Section 3.2.1
3.	Name of the SQL server	In this application example: SQLEXPRESS
4.	Name of the SQL server database	One SQL server can contain several databases. Use this parameter to specify which database you wish to connect to.

2.4.2 Transmitting SQL Commands

Formulate an SQL command and store it at the "command" input of the FB "LSqlMicrosoft". If the controller has established a connection to the SQL server, you can transfer the SQL command to the SQL server with a positive edge at the "executeCommand" input.

In the data block "SqlCommands" of this application example, a number of ready-made SQL commands can be found. You can use it.

The following figure shows the ready-made SQL commands:

Figure 2-9

SqlCommands

	Name	Data ty...	Start value
1	Static		
2	currentSqlCommand	USInt	6
3	sqlCommands	Array[0...	
4	sqlCommands[0]	String	'insert into PLCDATA_1 values (7,8, 9)'
5	sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
6	sqlCommands[2]	String	'Select IntegerValue1 from PLCDATA_1'
7	sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
8	sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where color = $'red$''
9	sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')'
10	sqlCommands[6]	String	'SELECT TOP (3) * FROM PLCDATA_2 ORDER BY Fruit'
11	sqlCommands[7]	String	'execute myProcedureIn @input1=30, @input2=6, @input3=84'
12	sqlCommands[8]	String	'execute myProcedureSelect'
13	sqlCommands[9]	String	'declare @myout1 char(30), @myout2 int execute myProcedureInOut @in1=$'pear$', @in2=$'yellow$', @in3=3, @out1=@myout1 OUTPUT, @out2=@myout...
14	sqlCommands[10]	String	'INSERT INTO [dbo].[errors] ([Text]) VALUES ($'remotetest$')'
15	sqlCommands[11]	String	'SELECT TOP (10) ValueReal1, ValueReal2, ValueReal3, ValueReal4, ValueReal5, ValueReal6, ValueReal7, ValueReal8, ValueReal9 FROM TestJulian'
16	sqlCommands[12]	String	'SELECT * FROM KompressionLog ORDER BY StentNr'
17	sqlCommands[13]	String	'execute SelectProc1'
18	sqlCommands[14]	String	'insert into TestDateTime values ($'01.01.1990 12:00:00$')'
19	sqlCommands[15]	String	'exec SetDateTime @Input1 = $'07.30.1990 12:00:00$''
20	sqlCommands[16]	String	'SELECT TOP (100) MesId, MesId2, MesId3 FROM TestJulian'
21	sqlCommands[17]	String	'SELECT TOP (1) DateTime FROM TestDateTime'
22	sqlCommands[18]	String	'SELECT TOP 2 * FROM FunctionTest ORDER BY ID'
23	sqlCommands[19]	String	'INSERT INTO [dbo].[FunctionTest] DEFAULT VALUES'
24	sqlCommands[20]	String	'use [SQLFromPLC] declare @myout1 bit execute NachtragDatensaetze @inDateiPfad= $'C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL...
25	sqlCommands[21]	String	'UPDATE [dbo].[KompressionLog] SET LotNummer = $'Test234$' WHERE KammerId = $'3$''

Note The example only supports the standard ASCII encoding.

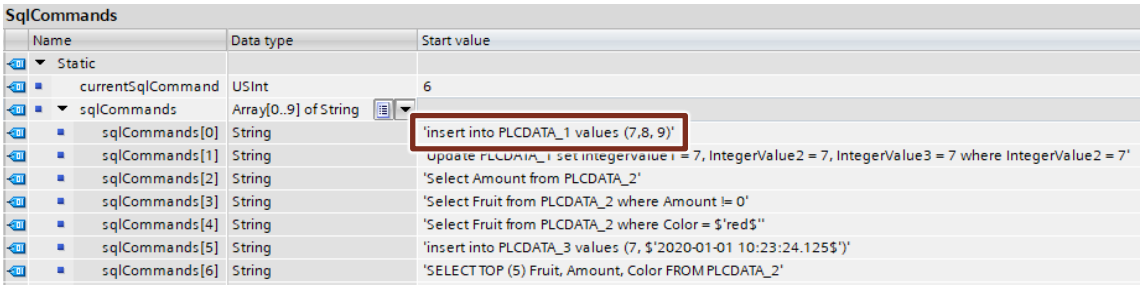
2.4.2.1 SQL command "insert into" – example integer values

SQL command

The following Figure shows an "insert into" SQL command for adding a new row containing integer values into a database table.

You can find examples of the SQL command "insert into" in the data block "SqlCommands":

Figure 2-10



This SQL command inserts a new row in the database table "PLCDATA_1" (see Section 2.2). Values (7,8,9) specifies the values that will be entered in the new row of the database table "PLCDATA_1".

- First column (IntegerValue1): 7
- Second column (IntegerValue2): 8
- Third column (IntegerValue3): 9

Executing the SQL command

A positive edge at the "executeCommand" input sends the SQL command to the database.

Result

The following Figure shows the contents of the table "PLCDATA_1" after this SQL command is executed.

Figure 2-11

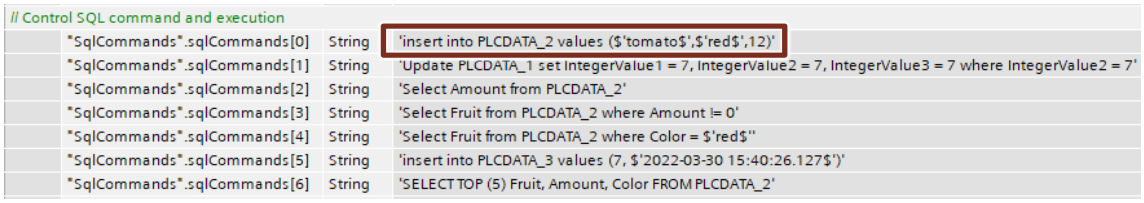
	IntegerValue1	IntegerValue2	IntegerValue3
1	7	8	9

2.4.2.2 SQL command "insert into" – example character strings

SQL command

The following Figure shows an "insert into" SQL command for adding a new row containing character strings into a database table.
You can find examples of the SQL command "insert into" in the data block "SqlCommands":

Figure 2-12



// Control SQL command and execution			
	"SqlCommands".sqlCommands[0]	String	'insert into PLCDATA_2 values ($'tomato$',$'red$',12)'
	"SqlCommands".sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
	"SqlCommands".sqlCommands[2]	String	'Select Amount from PLCDATA_2'
	"SqlCommands".sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
	"SqlCommands".sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where Color = $'red$''
	"SqlCommands".sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2022-03-30 15:40:26.127$')'
	"SqlCommands".sqlCommands[6]	String	'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2'

This SQL command inserts a new row in the database table "PLCDATA_2" (see Section 2.2). Values ($'tomato$',$'red$',12) specifies the values that will be entered in the new row of the database table "PLCDATA_2".

- First column (Fruit): tomato
- Second column (Color): red
- Third column (Amount): 12

Executing the SQL command

A positive edge at the "executeCommand" input sends the SQL command to the database.

Result

The following Figure shows the contents of the table "PLCDATA_2" after this SQL command is executed.

Figure 2-13

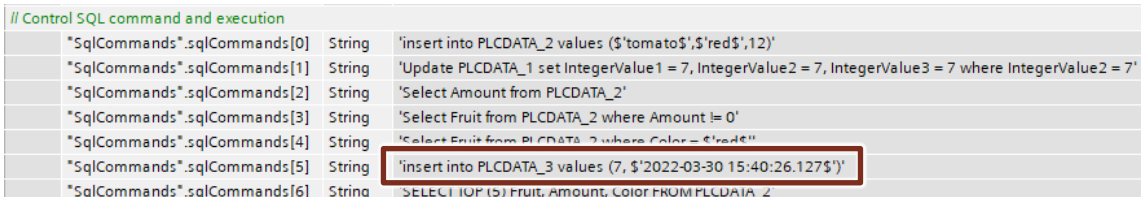
	Fruit	Color	Amount
1	tomato	red	12

2.4.2.3 SQL command "insert into" – example timestamps

SQL command

The following Figure shows an "insert into" SQL command for adding a new row containing timestamps into a database table.
You can find examples of the SQL command "insert" in the data block "SqlCommands":

Figure 2-14



This SQL command inserts a new row in the database table "PLCDATA_3" (see Section 2.2). Values (7,$'2022-03-30 15:40:26.125$') specifies the values that will be entered in the new row of the database table "PLCDATA_3".

- First column (Number): 7
- Second column (Occurrence): 2022-03-30 15:40:26.127

Executing the SQL command

A positive edge at the "executeCommand" input sends the SQL command to the database.

Result

The following Figure shows the contents of the table "PLCDATA_3" after this SQL command is executed.

Figure 2-15

	Number	Occurance
1	7	2022-03-30 15:40:26.127

2.4.2.4 The "select" SQL command

SQL command

The following Figure shows example of "select" SQL commands used to read values from a database table and perform further operations on them in the controller.

You can find examples of the SQL command "select" in the data block "SqlCommands":

Figure 2-16

SqlCommands

Name	Data type	Start value
Static		
currentSqlCommand	USInt	6
sqlCommands	Array[0..9] of String	
sqlCommands[0]	String	'insert into PLCDATA_1 values (7,8, 9)'
sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
sqlCommands[2]	String	'Select Amount from PLCDATA_2'
sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where Color = $'red$''
sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')'
sqlCommands[6]	String	'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2'

These SQL commands read values from the "Amount" or "Fruit" column of the database table "PLCDATA_2" (see Section 2.2).

Sample database table "PLCDATA_2"

The database table "PLCDATA_2" has the following content in this example:
Figure 2-17

Amount	Color	Fruit	Fresh	Country	CountryCode	City	Taste
10	Red	Apple	True	Germany	DE	Berlin	Good
3	Green	Pear	False	Italy	IT	Genua	No
2	Yellow	Banana	True	Africa	A	N.A.	Good
3	Red	Tomato	True	Netherlands	NE	Amsterdam	Well
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Changes

Below we will show you how the SQL "select" command works and what customizations you need to configure for your query.

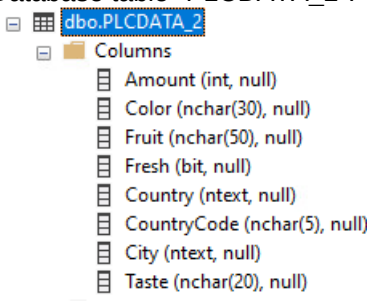
1. Adapt the PLC data type "typeTokenRow" to the database table from which the data is read with SQL command "select".
   In this application example the PLC data type "typeTokenRow" is adapted to the database table "PLCDATA_2".

   PLC data type "typeTokenRow":

typeTokenRow

	Name	Data type	Default value	Accessible f...
1	Amount	Int	0	
2	Color	String[30]	''	
3	Fruit	String[50]	''	
4	Fresh	Bool	false	
5	Country	String	''	
6	CountryCode	String[5]	''	
7	City	String	''	
8	Taste	String[20]	''	

   Database table "PLCDATA_2":



2. In the FB "DeserializeRows", examples for reading out character strings, integer values, and Boolean values can be found.
   In the query, match the column name to the column name of the database table you are reading.
   - The following figure shows the readout of integer values from the "Amount" column.

     Figure 2-18

```
IF #columns[#tempColumnLoopCounter].columnName = 'Amount' THEN
  REGION example INT
    //merge next two BYTE to WORD and then convert to INT
    #rows[#tempRowLoopCounter].Amount := WORD_TO_INT("LGF_MergeBytesToWord"(byte0 := #tdsTelegramArray[#statByteAddressCounter],
                                                                           byte1 := #tdsTelegramArray[#statByteAddressCounter + 1]));
  END_REGION example Int
```

   - The following figure shows the readout of Boolean values from the "Fresh" column.

```
ELSIF #columns[#tempColumnLoopCounter].columnName = 'Fresh' THEN
  REGION example Bool
    //convert BYTE to BOOL
    #rows[#tempRowLoopCounter].Fresh := BYTE_TO_BOOL(#tdsTelegramArray[#statByteAddressCounter]);
  END_REGION example Bool
```

   - The following figure shows the readout of character strings from the "Color" column.

```
ELSIF #columns[#tempColumnLoopCounter].columnName = 'Color' THEN
  REGION example String
    //move Bytes to temporary array
    MOVE_BLK(IN    := #tdsTelegramArray[#statByteAddressCounter], ...);

    //convert BYTE array to STRING
    #rows[#tempRowLoopCounter].Color := "LSql_BytesToString"(columnType := #columns[#tempColumnLoopCounter].columnType,
                                                             byteArray := #tempByteCharArray,
                                                             numberOfBytes := UDINT_TO_UINT(#tempValueLength));
  END_REGION example String
```

   - The following figure shows the readout of character strings from the "Fruit" column.

```
ELSIF #columns[#tempColumnLoopCounter].columnName = 'Fruit' THEN
  REGION example String
    //move Bytes to temporary array
    MOVE_BLK(IN    := #tdsTelegramArray[#statByteAddressCounter], ...);

    //convert BYTE array to STRING
    #rows[#tempRowLoopCounter].Fruit := "LSql_BytesToString"(columnType := #columns[#tempColumnLoopCounter].columnType,
                                                             byteArray := #tempByteCharArray, numberOfBytes := UDINT_TO_UINT(#tempValueL
  END_REGION example String
```

3. Insert any additional queries as needed.

Executing the SQL command

A positive edge at the "executeCommand" input sends the SQL command to the database.

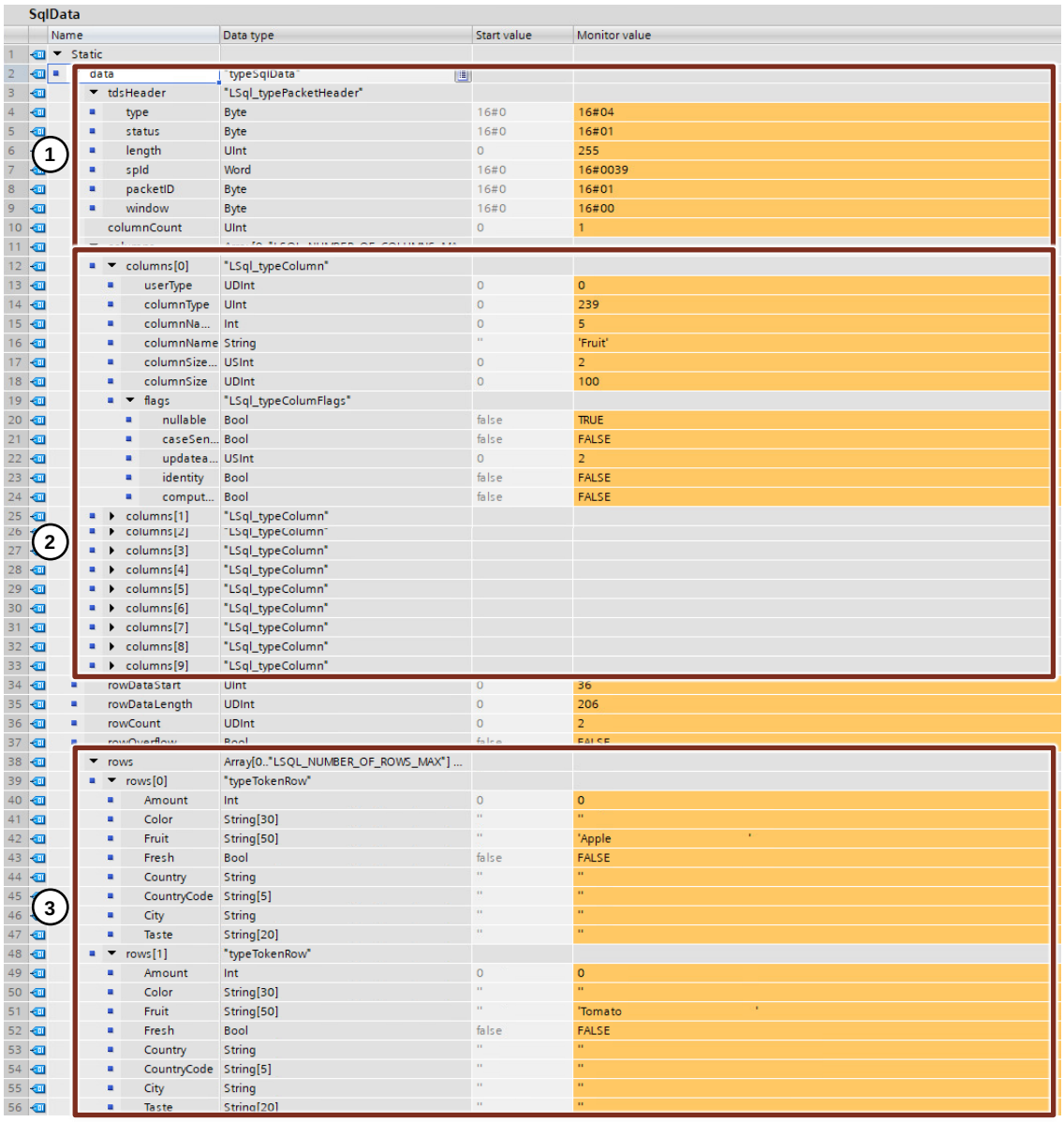
Result

The read-out data is contained in the TDS response telegram. The TDS response telegram is stored in the data block (DB) "SqlTdsReceive" in the data structure "tdsTelegramArray".

The FBs "AnalyzeTokens" and "DeserializeRows" prepare the TDS response telegram so that the data can be read by the user.

The prepared data are stored in the DB "SqlData" in the PLC data type structure "typeSqlData".

Figure 2-19



In the following table, a description of the read-out data can be found.

Table 2-15

No.	Description of read-out data
1.	TDS header
2.	Metadata of the SQL columns • Number of columns • Properties of the SQL columns
3.	Values from the read rows

The values of the read rows are contained in the individual elements of the "rows[x]" array.

The result of the SQL command "select Fruit from PLCDATA_2 where Color=$'red$'" can be seen in the Figure below.

Figure 2-20

SqlData

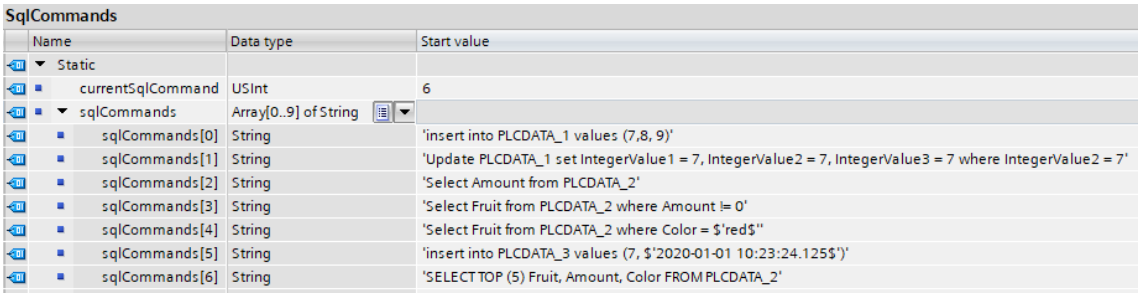
	Name	Data type	Start value	Monitor value
1	Static			
2	data	"typeSqlData"		
3	tdsHeader	"LSql_typePacketHeader"		
4	columnCount	UInt	0	1
5	columns	Array[0.."LSQL_NUMBER_OF_COLUMNS_MA...		
6	rowDataStart	UInt	0	36
7	rowDataLength	UDInt	0	206
8	rowCount	UDInt	0	2
9	rowOverflow	Bool	false	FALSE
10	rows	Array[0.."LSQL_NUMBER_OF_ROWS_MAX"] ...		
11	rows[0]	"typeTokenRow"		
12	Amount	Int	0	0
13	Color	String[30]	''	''
14	Fruit	String[50]	''	'Apple
15	Fresh	Bool	false	FALSE
16	Country	String	''	''
17	CountryCode	String[5]	''	''
18	City	String	''	''
19	Taste	String[20]	''	''
20	rows[1]	"typeTokenRow"		
21	Amount	Int	0	0
22	Color	String[30]	''	''
23	Fruit	String[50]	''	'Tomato
24	Fresh	Bool	false	FALSE
25	Country	String	''	''
26	CountryCode	String[5]	''	''
27	City	String	''	''
28	Taste	String[20]	''	''

Amount	Color	Fruit	Fresh	Country	CountryCode	City	Taste
10	Red ...	Apple ...	True	Germany	DE	Berlin	Good
3	Green ...	Pear ...	False	Italy	IT	Genua	No
2	Yellow ...	Banana ...	True	Africa	A	N.A.	Good
3	Red ...	Tomato ...	True	Netherlands	NE	Amsterdam	Well
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

2.4.2.5 Additional SQL commands

More examples of SQL commands in the "SqlCommands" data block can be found in the Figure below.

Figure 2-21



SqlCommands

	Name	Data type	Start value
	Static		
	currentSqlCommand	USInt	6
	sqlCommands	Array[0..9] of String	
	sqlCommands[0]	String	'insert into PLCDATA_1 values (7,8, 9)'
	sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
	sqlCommands[2]	String	'Select Amount from PLCDATA_2'
	sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
	sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where Color = $'red$''
	sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')'
	sqlCommands[6]	String	'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2'

Note In Section 3.4, information on calling stored procedures can be found.

2.5 Troubleshooting

2.5.1 Troubleshooting the FBs

If one of the FBs has an error, evaluate the output parameters "error" and "status". If the "diagnostics" output parameter is present, it should be evaluated along with the "error" and "status" output parameters.

"Error = TRUE" signals that an error occurred while the FB was working. The "status" provides unambiguous information on the status of the block. "diagnostics" gives you detailed status and diagnostic information from subfunctions that the FB uses internally.

Per the status concept of the SIMATIC programming style guide used here, the parameters "error" and the most significant bit (MSB) of "status" (bit 15) are identical. The remaining bits are used for an error code which points unambiguously to the cause. The error codes are also stored as constants in the local data of the block.

FB "LSql_Microsoft"

The Table below shows the status and error messages of the "LSql_Microsoft" FB.

Table 2-16

Error message/name of the constant in the FB	Value	Description
STATUS_NO_CALL	16#7000	No job is currently being processed.
STATUS_FIRST_CALL	16#7001	First call after incoming new job (rising edge at parameter "enable").
STATUS_SUBSEQUENT_CALL	16#7002	Subsequent call during active processing without additional information.
STATUS_TDISCON_SUCCESSFULL	16#7011	TDISCON called successfully.
ERR_UNDEFINED_STATE	16#8900	Error due to an undefined state in the state machine.
ERR_IN_BLOCK_OPERATION	16#8001	Incorrect command of the function block
ERR_PARAMETERIZATION	16#8200	Error during parameterization
ERR_PROCESSING_EXTERNAL	16#8400	External error, e.g., incorrect I/O signal
ERR_PROCESSING_INTERNAL	16#8600	Internal error, e.g., while calling a system module
ERR_AREA_RESERVED	16#8800	Error due to reserved area
ERR_USER_DEFINED_CLASSES	16#9000	User-specific error class
ERR_DISCONNECT	16#8601	Error when calling TDISCON.
ERR_CONNECT	16#8602	Error when calling TCON.
ERR_PRELOGIN	16#8603	Error in the execution of the pre-login.
ERR_PRELOGIN_DATA	16#8607	Error in converting the data for pre-login.
ERR_LOGIN	16#8604	Error while logging into the application.
ERR_LOGIN_DATA	16#8608	Error converting the login data.
ERR_TRCV	16#8605	Error when calling TRCV.
ERR_ARCHIVE	16#8606	Error while archiving in DB
ERR_SQLBATCH_DATA	16#8609	Error converting the data for the SQL command.
ERR_SQLBATCH_SEND	16#8610	Error sending SQL data.

FB "AnalyzeTokens"

The following table shows the status and error messages of the FB "AnalyzeTokens".

Table 2-17

Error message/name of the constant in the FB	Value	Description
ERR_UNKNOWNTOKEN	16#8601	Wrong type for the token stream "ColumnMetaData".
ERR_HEADER_INCORRECTLENGTH	16#8602	Wrong length for the token stream "ColumnMetaData".
ERR_COLUMNS_SIZENOTDEFINED	16#8603	At least one column size is not defined.
ERR_COLUMNS_TYPENOTDEFINED	16#8604	At least one column type is undefined or unknown.
ERR_COLUMNS_OVERFLOW	16#8605	Column overflow
ERR_TOKENDONE_UNKNOWNSTATUS	16#8606	The DONE token has an unknown status.
ERR_TOKENDONE_WRONGTOKENTYPE	16#8607	The DONE token has the wrong token type.
ERR_HEADER_WRONGSTATUS	16#8608	The TDS header has an incorrect status.
ERR_TOKENROW_NOMETADATA	16#8610	No "ColumnMetaData" were decoded.
ERRLENGTHINCORRECT	16#8611	The length of the analyzed data does not match the total TDS length.
ERR_TOKENERROR	16#8609	An ERROR token was found

FB "DeserializeRows"

The Table below shows the status and error messages of the "DeserializeRows" FB.

Table 2-18

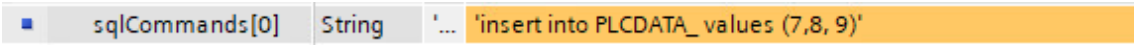
Error message/name of the constant in the FB	Value	Description
ERR_ROWARRAY_SIZENOTDEFINED	16#8601	Size of the data array "rows" is not defined.
ERR_ROWARRAY_SIZEINCORRECT	16#8002	Size of the data array "rows" is not correct.
ERR_ROWDATA_LENGTHNOTDEFINED	16#8603	The length of the token "Row data" is not defined.
ERR_COLUMNS_COUNTINCORRECT	16#8604	Number of columns is incorrect.
ERR_COLUMNS_COLUMNNOTEXISTING	16#8605	At least one column name cannot be processed.

2.5.2 ERROR Token

If an error occurs when trying to log into the SQL server or when processing an SQL command, the SQL server sends an ERROR token in its response telegram. The ERROR token contains, among other things, information about the error code and the error message.
The FB "AnalyzeTokens" evaluates received tokens. If the block detects an ERROR token, it provides the information contained at the "errorToken" output.

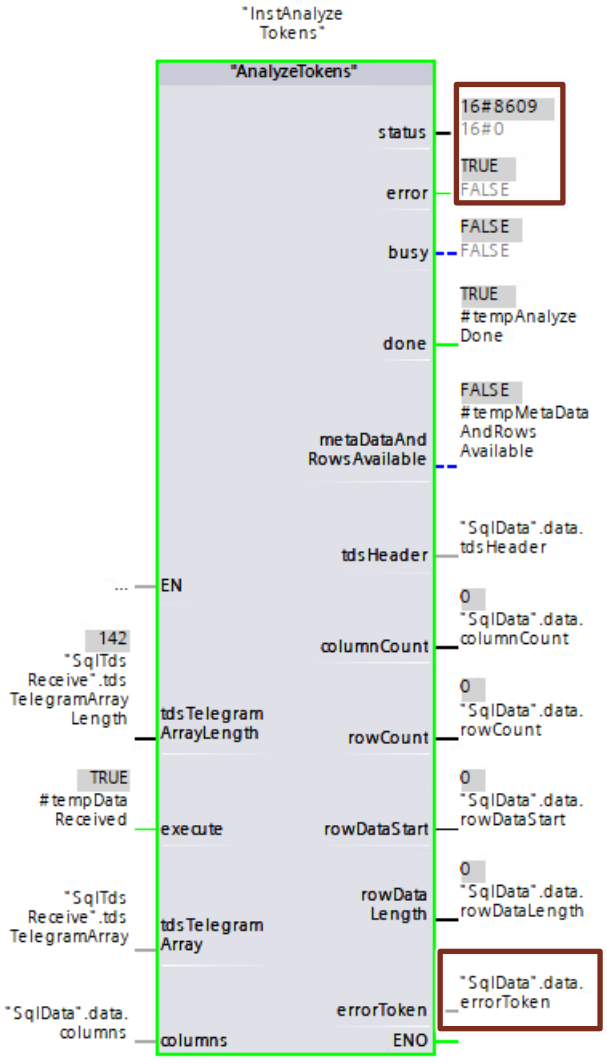
In the following example, the SQL command "insert" was executed on a non-existent database table:

Figure 2-22



The SQL server reports an error in its response telegram. The FB "AnalyzeTokens" evaluates the response telegram. The block detects the ERROR token, sets the error bit and the status "16#8609" ("ERROR token detected").

Figure 2-23



The content of the ERROR token is made available to you at the "errorToken" output.

Figure 2-24

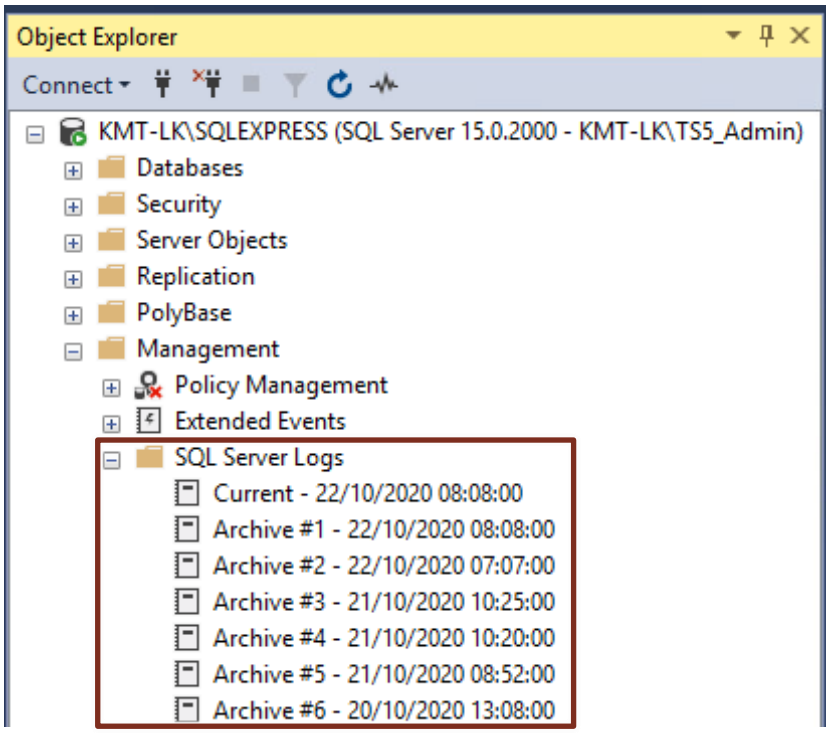
SqlData

Name	Data type	Start value	Monitor value
Static			
data	"typeSqlData"		
tdsHeader	"LSql_typePacketHeader"		
columnCount	UInt	0	0
columns	Array[0.."LSQL_NUMBER_OF_COLUMNS_MA...		
rowDataStart	UInt	0	0
rowDataLength	UDInt	0	0
rowCount	UDInt	0	0
rowOverflow	Bool	false	FALSE
rows	Array[0.."LSQL_NUMBER_OF_ROWS_MAX"] ...		
errorToken	"LSql_typeErrorToken"		
TokenLength	UInt	0	118
SQLErrorNumber	UDInt	0	208
State	Byte	16#0	16#01
Class	USInt	0	16
ErrorMessageL...	UInt	0	31
ErrorMessage	String	''	'Invalid object name $'PLCDATA_$'.'
ServerNameLe...	USInt	0	21
ServerName	String	''	'WIN10-20H2\SQLEXPRESS'
ProcNameLeng...	USInt	0	0
ProcName	String	''	''
LineNumber	UDInt	0	1

2.5.3 Status and Error Messages from the SQL Server

You can perform a detailed error analysis directly in "SQL Server Management Studio". To do this, go to the folder "Management > SQL Server Logs". There you can find the server logs where, among other things, error messages are stored.

Figure 2-25



3 Useful Information

3.1 Fundamentals of Microsoft SQL Server 2019 Express

Microsoft SQL Server 2019 Express

Microsoft SQL Server is a powerful database management system for SQL databases. The free Express version is designed for desktop and server applications. It supports up to 10 gigabytes of storage per database.

You can download SQL Server 2019 Express from the following link:

https://www.microsoft.com/en-us/SQL Server/SQL Server-downloads

Microsoft SQL Server Configuration Manager

SQL Server Configuration Manager is a tool for managing the services associated with SQL server, configuring the network protocols used by SQL server and managing the configuration of network connectivity of SQL server client computers. SQL Server Configuration Manager is installed together with SQL server. Because the SQL Server Configuration Manager is a snap-in for the Microsoft Management Console program and not a standalone program, the SQL Server Configuration Manager does not appear as an application in newer versions of Windows.

To open the SQL Server Configuration Manager, navigate to the file location "C:\Windows\SysWOW64" and click the file "SQLServerManager<Version>.msc".

Microsoft SQL Server Management Studio

The free Microsoft SQL Server Management Studio provides tools for configuring, monitoring, and managing instances of SQL servers and databases. It makes it possible to send queries and scripts to databases in the form of SQL commands. In this way you can enter new data to the database table, or read existing data.

You can download Microsoft SQL Server Management Studio from the following link:

https://docs.microsoft.com/en-us/sql/ssms/download-SQL Server-management-studio-ssms

TDS – Tabular Data Stream Protocol

The Tabular Data Stream protocol is a protocol on the application layer (layer 7) of the ISO/OSI reference model. It enables interaction with a Microsoft SQL Server including authentication and encryption of communication. After successful login to the SQL server, SQL commands can be exchanged with the server's databases using this protocol. Data is transported over TCP/IP.

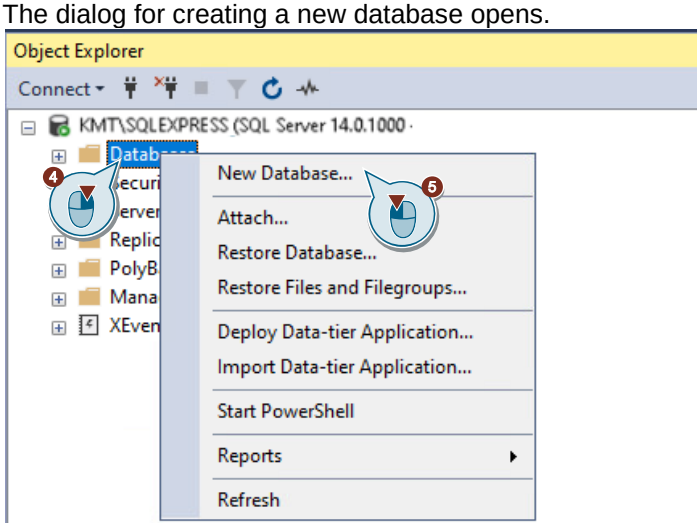
The Tabular Data Stream is described in detail in the Microsoft Technical Documentation: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-tds/b46a581a-39de-4745-b076-ec4dbb7d13ec

3.2 Microsoft SQL Server 2019 Express Setup

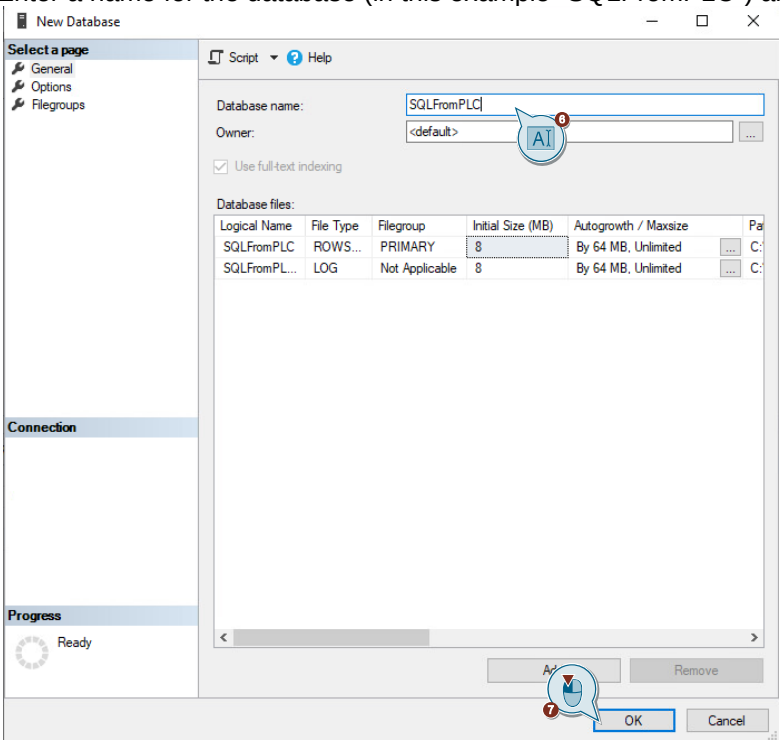
3.2.1 Creating the Database and Tables

Database

1. Launch Microsoft SQL Server Management Studio.
2. Sign in to the Microsoft SQL Server Express database with the Windows authentication mode.
3. Expand the SQLEXPRESS instance.
4. Right-click the "Database" menu.
   The context menu opens.
5. Click the "New Database…" button.
   The dialog for creating a new database opens.



6. Enter a name for the database (in this example "SQLFromPLC") and click "OK".

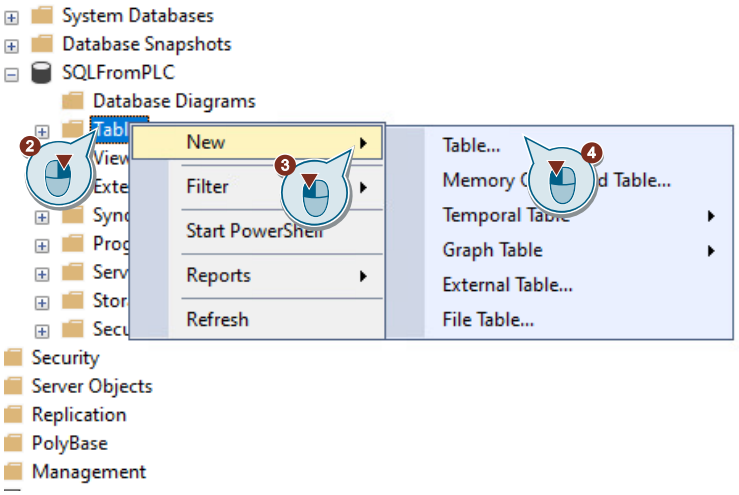


The database is created in the "Database" folder of the "Object Explorer".

Tables

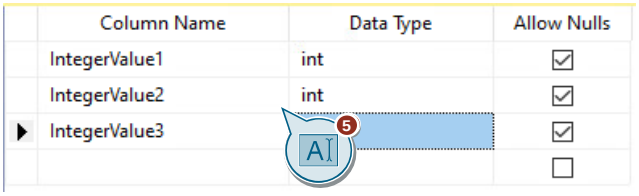
Now create tables in the database.

1. Expand the new database.
2. Right-click "Tables".
   The context menu opens.
3. Click the arrow symbol next to "New".
   The context menu is extended.
4. Click "Table…".

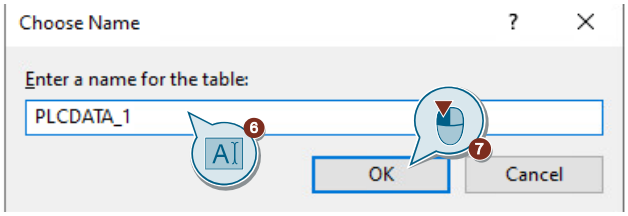


The table creation editor appears.

5. Define the columns and the values. To save the table, press the key combination <CTRL + S>.



6. Assign a name to the table (e.g., "PLCDATA_1").
7. Click "OK".



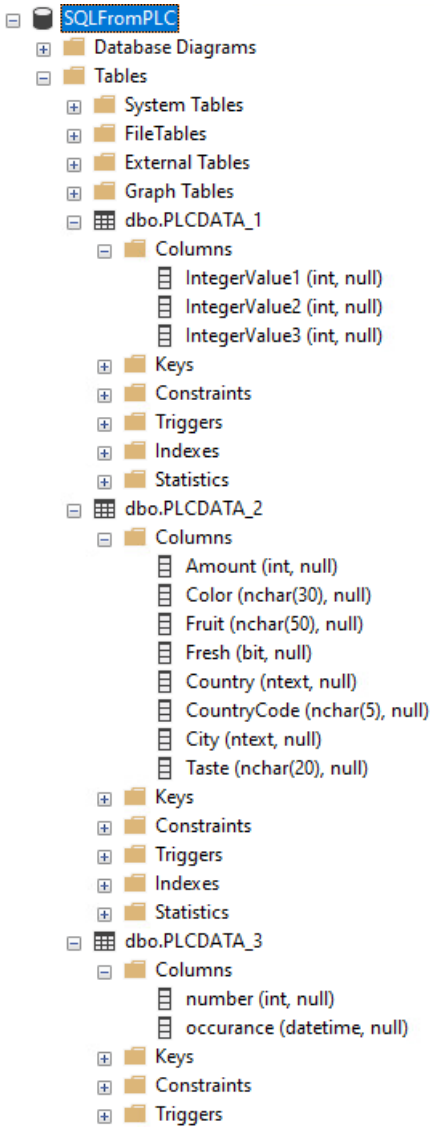
The table is created in the folder "Database > Table" in the "Object Explorer".

8. Create more tables.

Database of this example

The following Figure shows you the database of this example in SQL Server Management Studio.

Figure 3-1

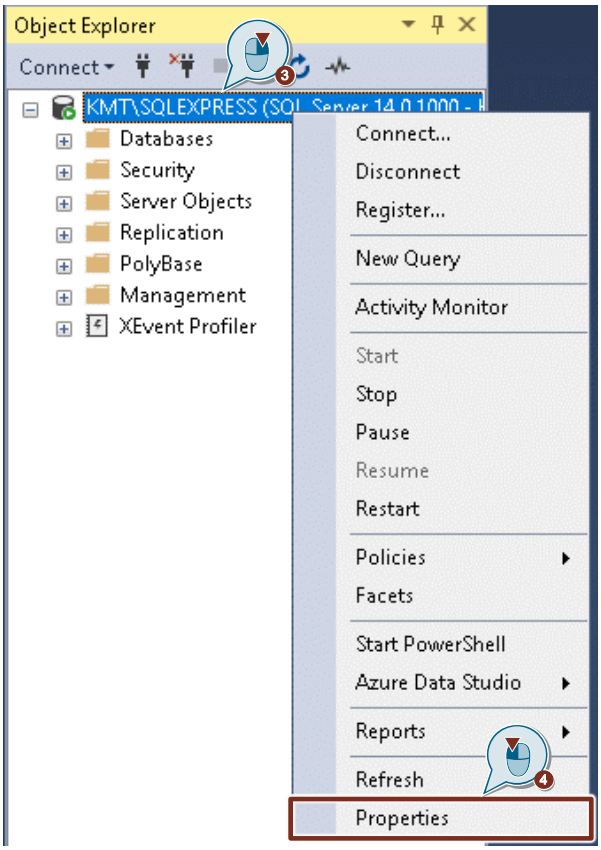


3.2.2 Logging into to the SQL Server

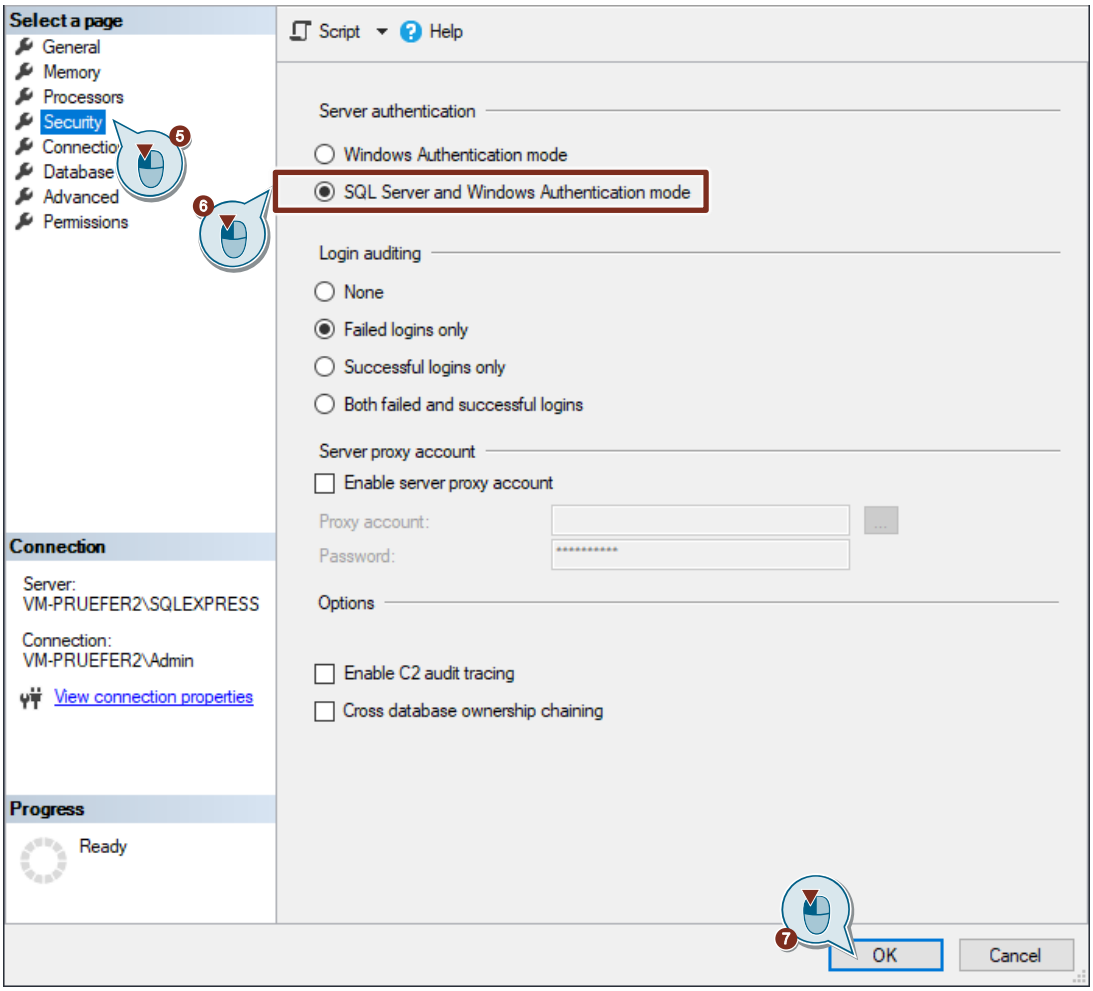
Login to a Microsoft SQL Server Express database via the SQL server authentication mode must be allowed in the database settings. This is required in order to log in to the database with username and password via the TDS protocol.

Enabling the SQL Server Authentication Mode

1. Launch Microsoft SQL Server Management Studio.
2. Sign in to the Microsoft SQL Server Express database with the Windows authentication mode.
3. Right-click on the SQLEXPRESS instance.
   The context menu opens.
4. Click on "Properties".
   The Properties dialog for the SQLEXPRESS instance opens.

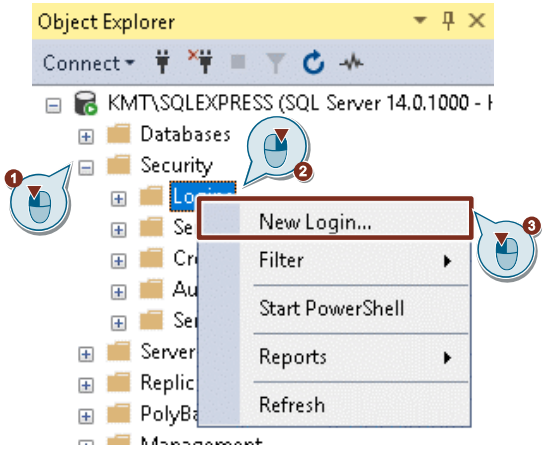


5. Select the "Security" page.
6. Enable the option "SQL Server and Windows Authentication mode" for server authentication.
7. Click the "OK" button to apply the setting.

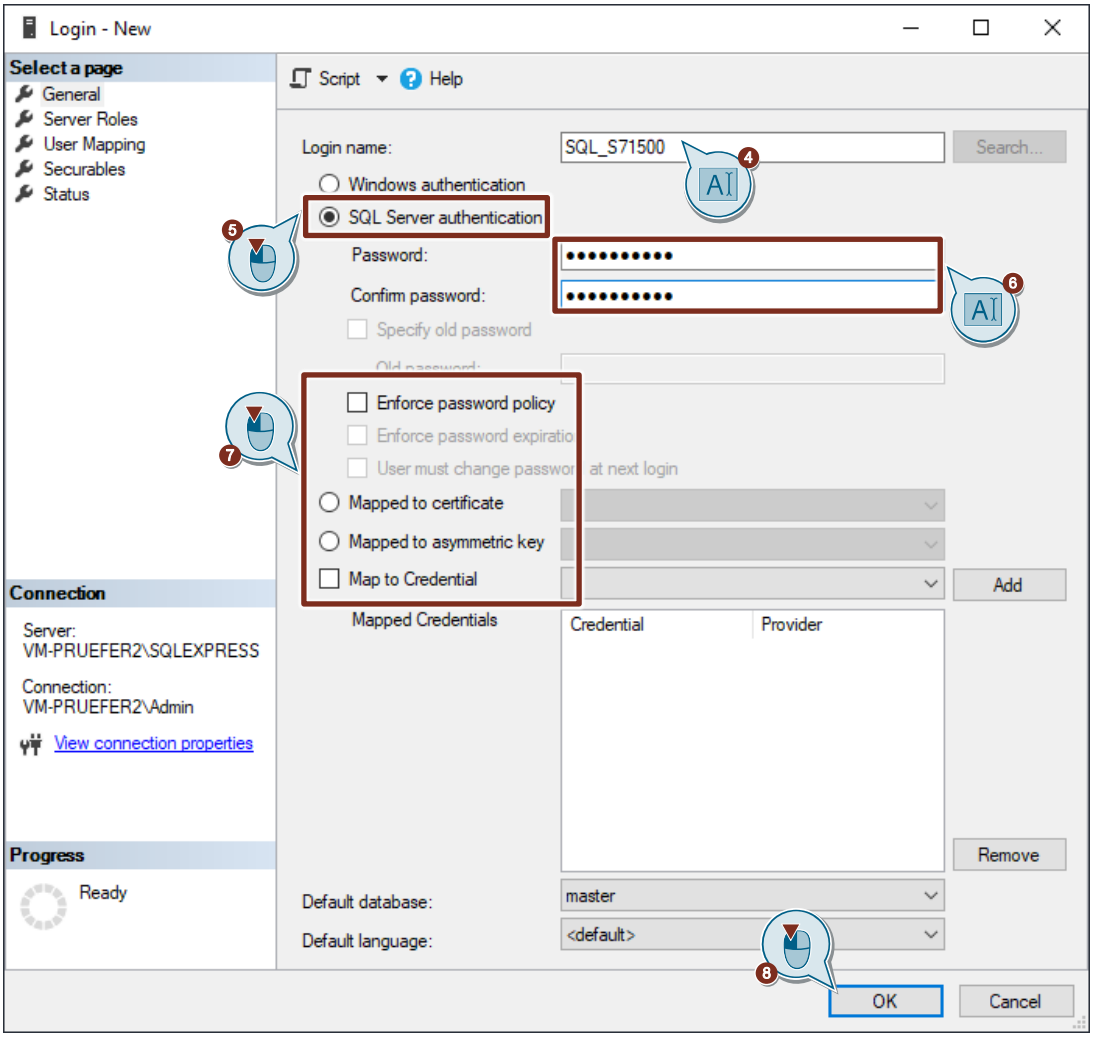


Create users

1. In the "Object Explorer", open the "Security" folder.
2. Right-click on the "Logins" folder.
   The context menu opens.
3. Click "New Login".
   The "Login - New" dialog opens.



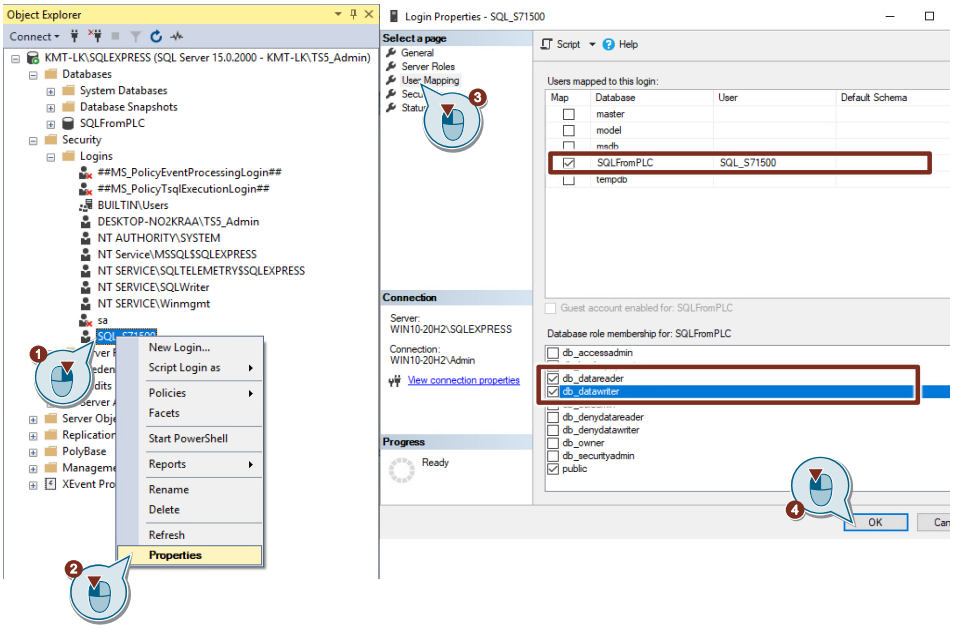
4. Enter a login name.
5. Select the option "SQL Server authentication".
6. Enter a password and confirm the password.
7. Deactivate the additional options.
8. Click the "OK" button to apply the setting.



The user has been created in the "Object Explorer" in the "Security > Logins" folder.

Assign user permissions

1. Open the folder "Security >Login" in the "Object Explorer" and right-click the user you just created.
2. Select "Properties" in the context menu.
3. Open the "User Mapping" page in the Properties. Assign the "SQLFromPLC" database to the user. Assign the "db_datareader" and "db_datawriter" roles to the user.
4. Click the "OK" button to apply the setting.



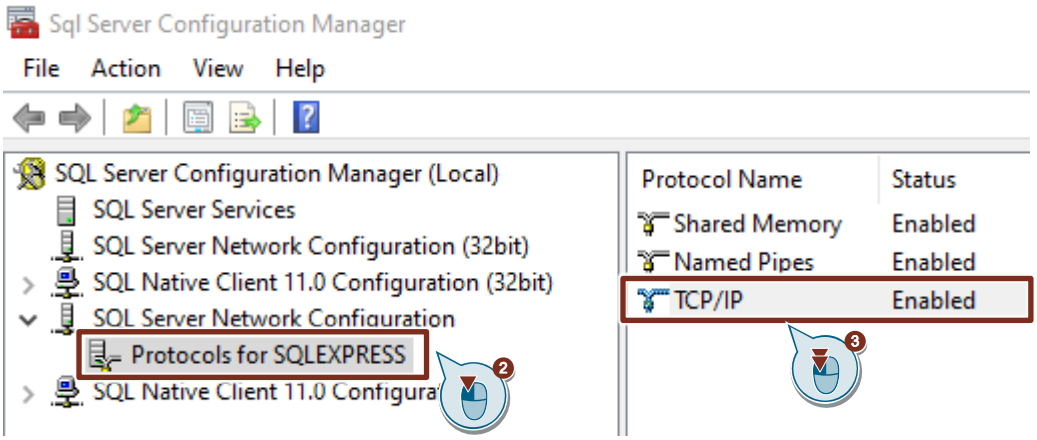
3.2.3 Port Sharing in the SQL Server

Note Port 1433 is the default port for Microsoft SQL Server databases.

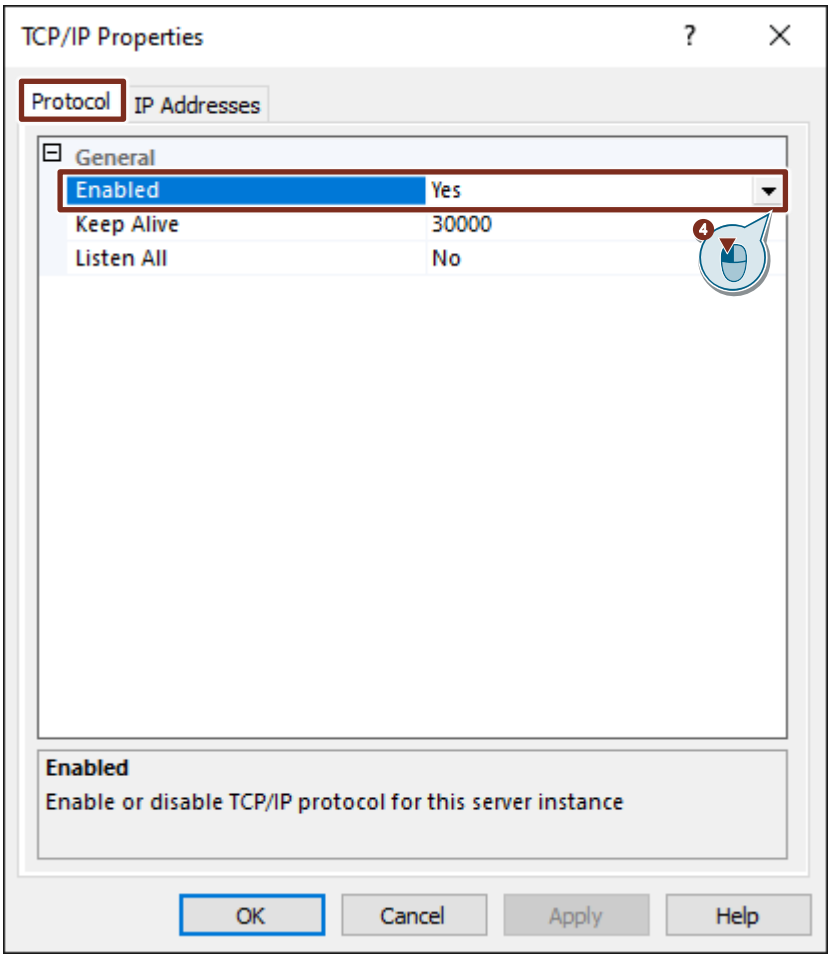
Note If the firewall is enabled on the PC with the Microsoft SQL Server database, TCP port "1433" must be enabled for incoming connections in the firewall.

In order for the SQL server to be accessible on the network, it is necessary to set up port sharing in the SQL server.

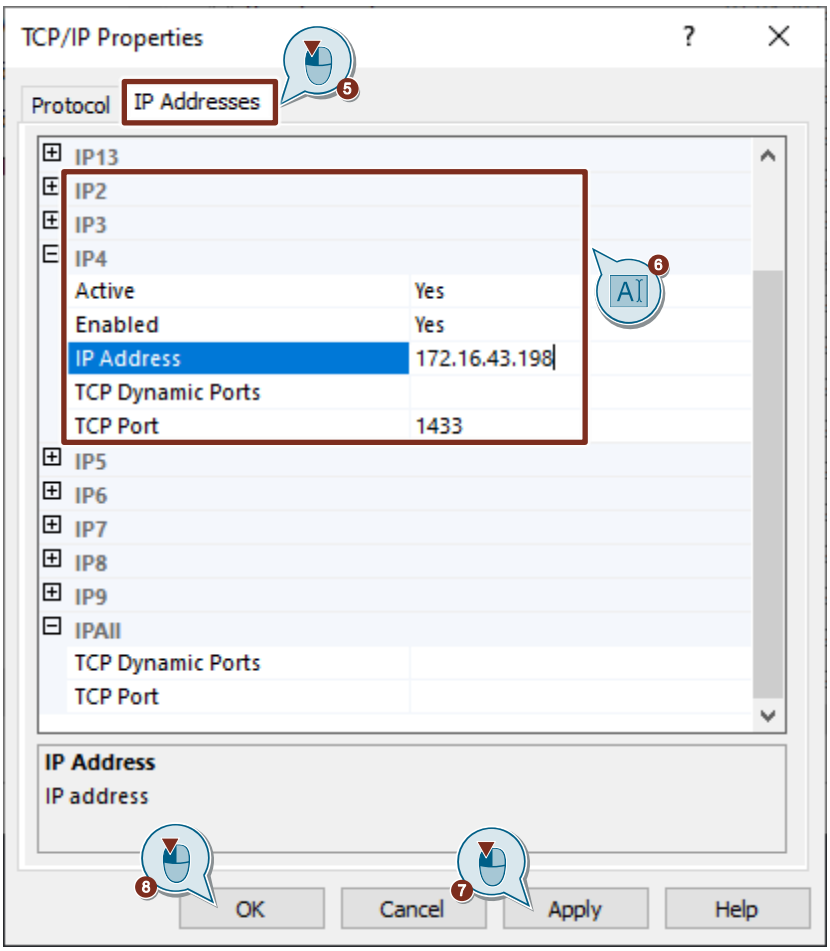
1. Launch "SQL Server Configuration Manager".
2. Navigate to "SQL Server Network Configuration > Protocols for SQLEXPRESS".
3. Double-click the "TCP/IP" protocol.
   The Properties dialog opens.



4. In the "Protocol" tab, enable the "TCP/IP" protocol.



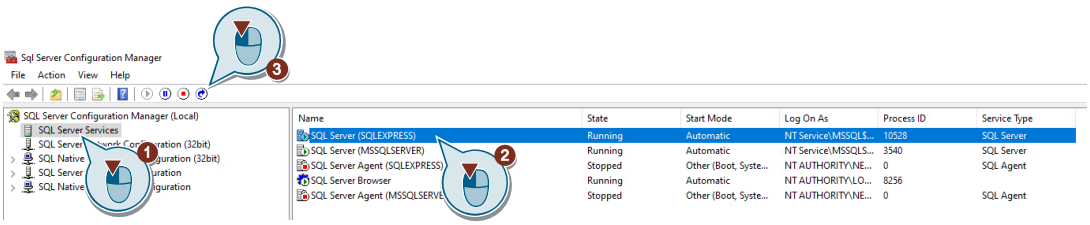
5. Open the "IP addresses" tab.
6. Configure the following settings in the "IP4" area, for instance.
   - Enter the IP address of the network interface
   - Enter port 1433
   - Enable interface
7. Click "Apply" to apply your settings.
8. Click on the "OK" button to close the Properties dialog.



3.2.4 Restarting the Server

For the changes to take effect, restart the SQL server service.

1. Navigate to "SQL Server Services".
2. Select the entry "SQL Server (SQLEXPRESS)".
3. Restart the server using the corresponding icon.



3.2.5 Testing the Connection to the SQL Server

Telnet

From another PC, test that Telnet connects to the SQL server IP address and port.

Detailed information on Telnet can be found via the following link:

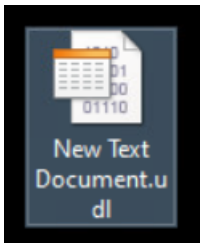
https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/telnet

UDL file

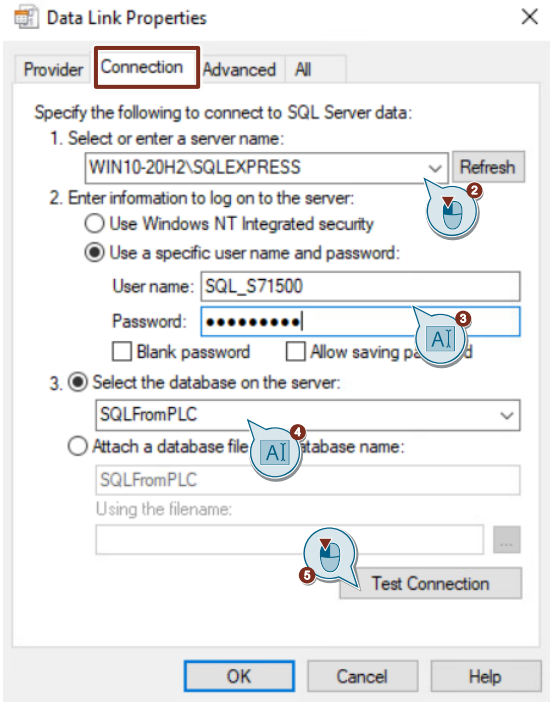
With a UDL file you can test the connection to the SQL server without using Telnet, a port scanner, or other software.

To create a UDL file, create a new text file on the desktop and change the extension of this file to .udl.

Figure 3-2

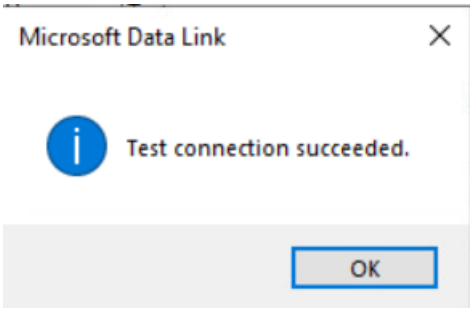


1. Double-click the file you just created. The Data Link Properties dialog opens. You are in the "Connections" tab.
2. Configure the server name.
3. Select the "Use a specific user name and password" option to enter a user ID that has the permissions for the entered SQL server.
4. Enter the name of the database.
5. To test the connection to SQL server, press the "Test Connection" button.



If the test was successful, it will be confirmed with the message "Test Connection succeeded".

Figure 3-3



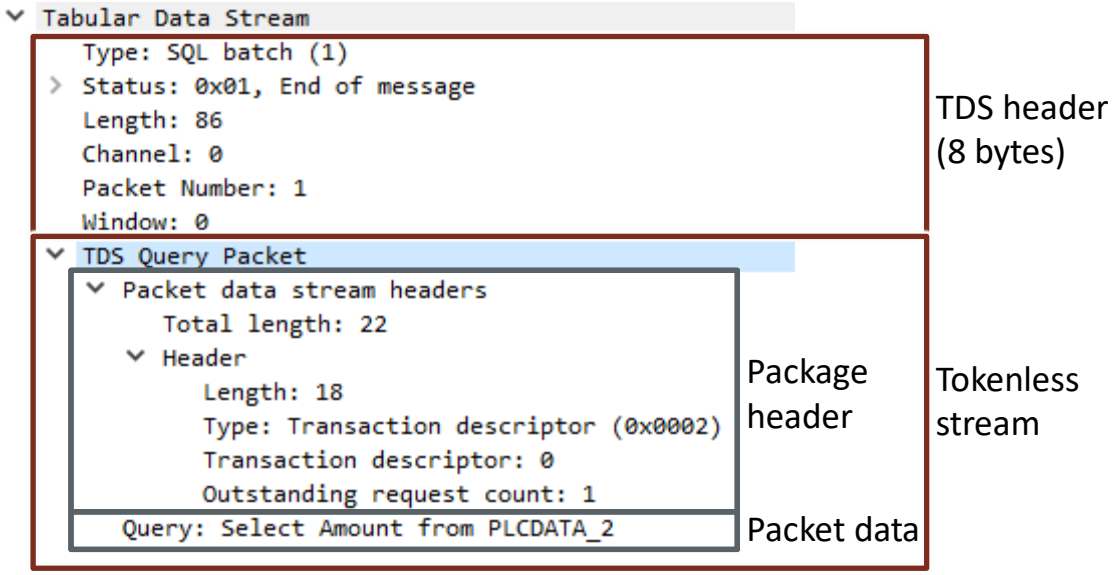
3.3 Tabular Data Stream (TDS) Structure

A Tabular Data Stream (TDS) comprises the following components.

- TDS header (8 bytes)
  Table 3-1 shows the structure of the TDS header.
- Packet data
  The messages in the packet data can be one of the following types:
  - Token stream
    A token stream consists of one or more tokens, each followed by some token-specific data. A token is an identifier (1 byte) used to describe the data that follow it.
    The messages contained in the packet data of the TDS response telegram are of the "Token stream" type.
  - Tokenless stream
    The packet header of a tokenless stream contains all information required to describe the packet data.
    The messages contained in the packet data of the SQL batch are of the "Tokenless stream" type.
- DONE token (13 bytes) for a TDS response telegram
  The DONE token marks the end of the response for each SQL command that is executed.
  Table 3-2 shows the structure of the DONE token.

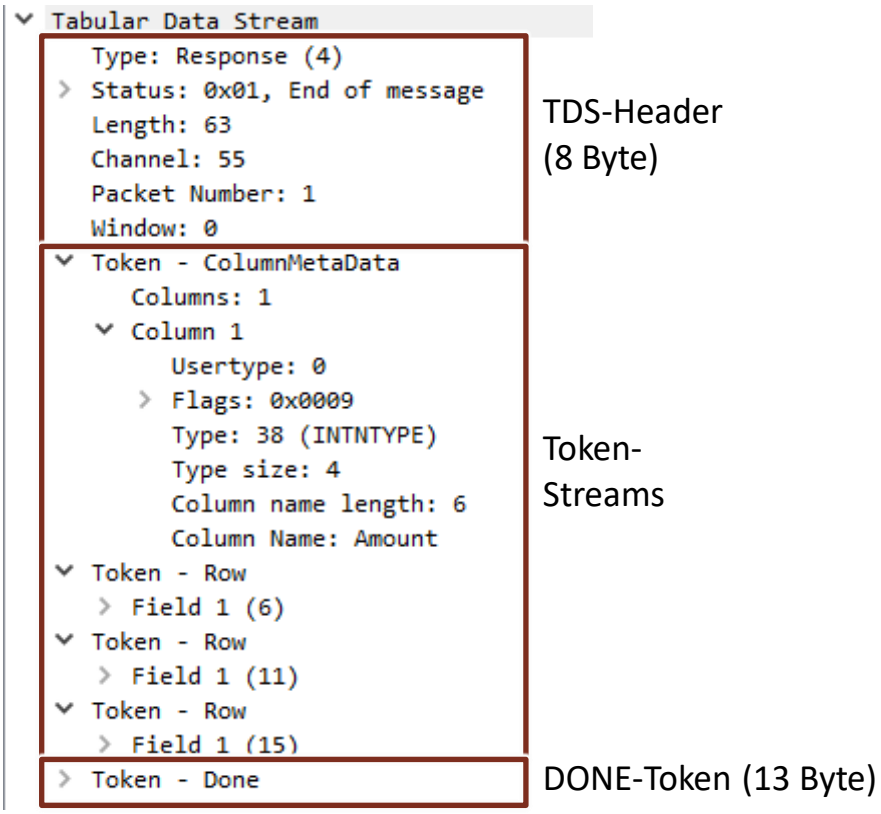
The following Figure shows a SQL batch.

Figure 3-4



The following Figure shows a TDS response telegram.

Figure 3-5



The following Table shows the structure of the TDS header.

Table 3-1

TDS header	Description
Type	Message type 1 = SQL batch 4 = TDS response telegram from SQL server
Status	Message status 0 = "normal" message 1 = End of message (EOM) The EOM indicates the last packet in the message.
Length	Length of the tabular data stream.
SPID	Process ID on the server corresponding to the current connection.
PacketID	ID of a message for a package.
Window	This parameter is currently unused.

The following Table shows the structure of the DONE token.

Table 3-2

DONE token	Description
Type	DONE token: 0xFD
Status	Token status 0x10 = DONE_COUNT, i.e., the "DoneRowCount" value is valid.
CurCmd	The token of the current SQL command.
DoneRowCount	The number of rows affected by the SQL command. The value of "DoneRowCount" is only valid if the value of Status contains the value "0x10".

3.4 Executing Stored Procedures on the SQL Server

3.4.1 Overview

Complex "select" SQL commands can easily exceed the 254-character limit. If you wish to use queries longer than 254 characters, you have the ability to call a "Stored Procedure". This is the most high-performance way of executing a long query to the database.

A stored procedure is a function that works through the stored queries and then outputs the result to the user.

Detailed information on stored procedures can be found via the following link:

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/

3.4.2 Calling a Stored Procedure without Inputs and Outputs

Transact-SQL code of the stored procedure

The Figure below shows a stored procedure without inputs or outputs. It executes the following function:

- The SQL command "select" reads a row from the "Amount" column from the "PLCDATA_2" database table.

Figure 3-6

```
USE [SQLFromPLC]
GO
/****** Object:  StoredProcedure [dbo].[myProcedureSelect]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Name
-- Create date:
-- Description:
-- =============================================
CREATE PROCEDURE [dbo].[myProcedureSelect]
--@output1 int output
    -- Add the parameters for the stored procedure here
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here

    SELECT Amount from PLCDATA_2
--  set @output1 = 1
END
```

Name der Prozedur

To execute the stored procedure without inputs or outputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

execute <Name of procedure>

Reading integer values

The Figure below shows an example of a SQL command that executes a stored procedure without inputs and outputs.

Figure 3-7

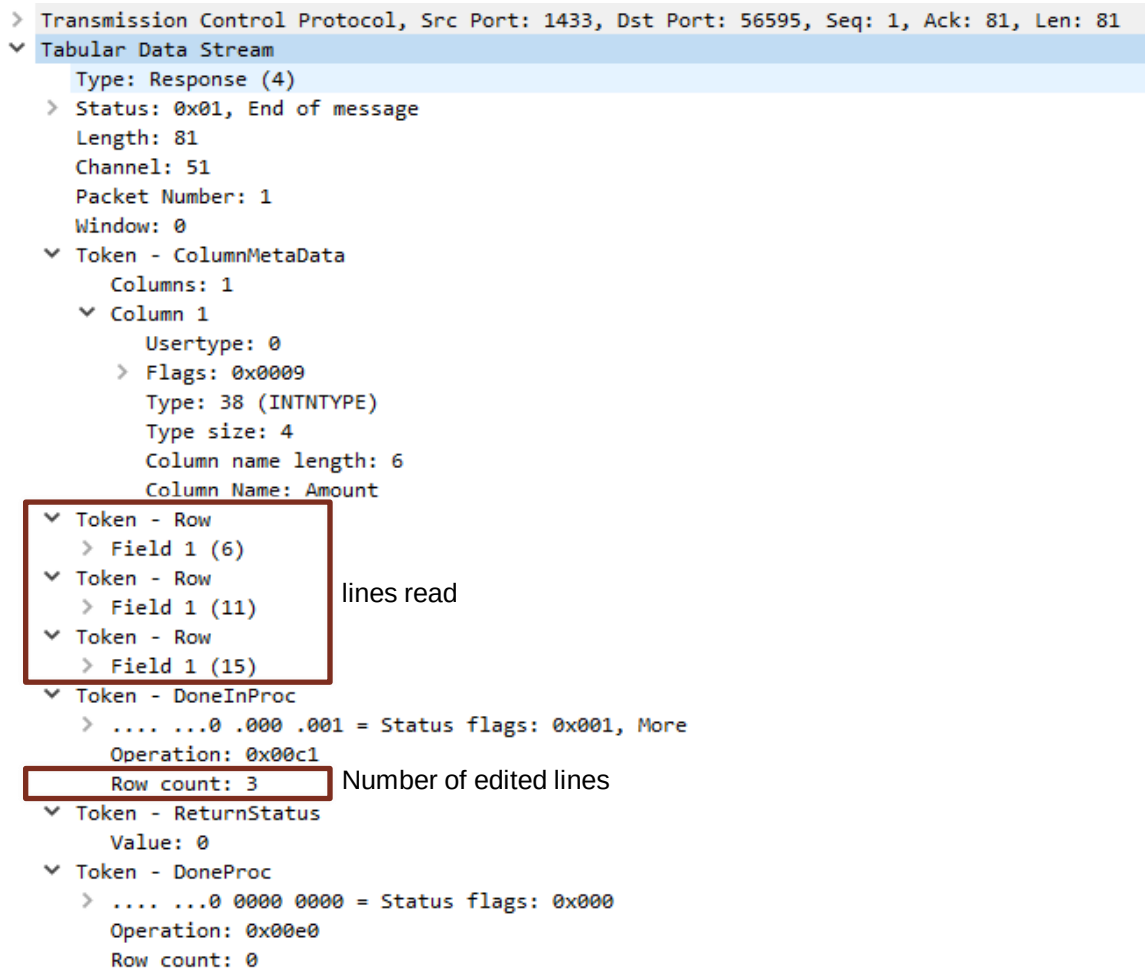
SqlCommands

	Name	Data type	Start value
	Static		
	currentSqlCommand	USInt	6
	sqlCommands	Array[0..9] of String	
	sqlCommands[0]	String	'insert into PLCDATA_1 values (7,8, 9)'
	sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
	sqlCommands[2]	String	'Select Amount from PLCDATA_2'
	sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
	sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where color = $'red$''
	sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')'
	sqlCommands[6]	String	'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2'
	sqlCommands[7]	String	'execute myProcedureIn @input1=30, @input2=6, @input3=84'
	sqlCommands[8]	String	'execute myProcedureSelect'

Result

The Figure below shows the TDS response telegram to the execution of the stored procedure without inputs and outputs.

Figure 3-8



3.4.3 Calling a Stored Procedure with Inputs

Transact-SQL code of the stored procedure

The Figure below shows a stored procedure with 3 inputs; it executes the following functions:

- The SQL command "insert into" adds a new row with values (passed at the inputs) to the database table "PLCDATA_1".

Figure 3-9

```
USE [SQLFromPLC]
GO
/****** Object:  StoredProcedure [dbo].[myProcedureIn]    Script Date: 07.04.2022 14:33:29
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Name
-- Create date:
-- Description:
-- =============================================
CREATE PROCEDURE [dbo].[myProcedureIn]
    -- Add the parameters for the stored procedure here
    @input1 int,
    @input2 int,
    @input3 int
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    Insert into PLCDATA_1 values (@input1, @input2, @input3)


END
```

Name of procedure

inputs

To execute the stored procedure with 3 inputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

execute <Name of procedure> @<Name of input 1> = value, @<Name of input 2> = value, @<Name of input 3> = value

Adding integer values

The Figure below shows an example of a SQL command that executes a stored procedure with 3 inputs.

Figure 3-10

SqlCommands

Name	Data type	Start value
Static		
currentSqlCommand	USInt	6
sqlCommands	Array[0..9] of String	
sqlCommands[0]	String	'insert into PLCDATA_1 values (7,8, 9)'
sqlCommands[1]	String	'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7'
sqlCommands[2]	String	'Select Amount from PLCDATA_2'
sqlCommands[3]	String	'Select Fruit from PLCDATA_2 where Amount != 0'
sqlCommands[4]	String	'Select Fruit from PLCDATA_2 where color = $'red$''
sqlCommands[5]	String	'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')'
sqlCommands[6]	String	'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2'
sqlCommands[7]	String	'execute myProcedureIn @input1=30, @input2=6, @input3=84'
sqlCommands[8]	String	'execute myProcedureSelect'

Result

The Figure below shows the TDS response telegram to the execution of the stored procedure with 3 inputs.

Figure 3-11

```
> Transmission Control Protocol, Src Port: 1433, Dst Port: 60961, Seq: 1, Ack: 141, Len: 26
v Tabular Data Stream
    Type: Response (4)
  > Status: 0x01, End of message
    Length: 26
    Channel: 53
    Packet Number: 1
    Window: 0
  v Token - ReturnStatus
      Value: 0
  v Token - DoneProc
    > .... ...0 0000 0000 = Status flags: 0x000
      Operation: 0x00e0
      Row count: 0
```

3.4.4 Calling a Stored Procedure with Inputs and Outputs

Transact-SQL code of the stored procedure

The Figure below shows a stored procedure with 3 inputs and 2 outputs; it executes the following functions:

- The SQL command "insert into" adds a new row with values (passed at the inputs) to the database table "PLCDATA_2".
- The SQL command "select" reads a row from the "Amount" column from the "PLCDATA_2" database table.
- Values are assigned to the outputs.

Figure 3-12

```
USE [SQLFromPLC]
GO
/****** Object:  StoredProcedure [dbo].[myProcedureInOut]    Script Date: 08.04.2022 11:57:37
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Name
-- Create date:
-- Description:
-- =============================================
CREATE PROCEDURE [dbo].[myProcedureInOut]
    -- Add the parameters for the stored procedure here
    @in1 char(30),
    @in2 char(30),
    @in3 int,
    @out1 char(30) output,
    @out2 int output
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here

    insert into PLCDATA_2 values (@in1,@in2,@in3)
        Select Amount from PLCDATA_2 where Color='red'
        set @out1 = 'yellow'
        set @out2 = 6
END
```

Name of procedure

Inputs and outputs

To execute the stored procedure with inputs and outputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

declare @myout1 char(30), @myout2 int execute <Name of procedure> @<Name of input 1>=value, @<Name of input 2>=value, @<Name of input 3>=value, @<Name output 1>=@myout1 OUTPUT, @<Name of output 2>=@myout2 OUTPUT select @myout1 as $'myOut1$', @myOut2 as $'myOut2$'

Reading and inserting integer values

The Figure below shows an example of a SQL command that executes a stored procedure with 3 inputs and 2 outputs.

Figure 3-13

Result

The Figure below shows the TDS response telegram to the execution of the stored procedure with 3 inputs and 2 outputs.

Figure 3-14

```
Transmission Control Protocol, Src Port: 1433, Dst Port: 55541, Seq: 1, Ack: 411, Len: 183
Tabular Data Stream
   Type: Response (4)
 > Status: 0x01, End of message
   Length: 183
   Channel: 53
   Packet Number: 1
   Window: 0
 v Token - ColumnMetaData
      Columns: 1
    > Column 1
 v Token - Row
    > Field 1 (15)
 v Token - Row
    > Field 1 (22)
 v Token - Row
    > Field 1 (18)
 v Token - DoneInProc
    > .... ...0 .000 .001 = Status flags: 0x001, More
      Operation: 0x00c1
      Row count: 3
 v Token - ReturnStatus
      Value: 0
 v Token - DoneProc
    > .... ...0 0000 0001 = Status flags: 0x001, More
      Operation: 0x00e0
      Row count: 0
 v Token - ColumnMetaData
      Columns: 2
    > Column 1
    > Column 2
 v Token - Row
    > Field 1 (yellow                    )
    > Field 2 (6)
 v Token - Done
    > .... ...0 .001 0000 = Status flags: 0x010, Row count valid
      Operation: 0x00c1
      Row count: 1
```

lines read

Number of edited lines

Values assigned to the outputs

4 Appendix

4.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers
– ranging from basic support to individual support contracts. Please send queries to Technical Support via Web form:

siemens.com/SupportRequest

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that’s tailored to the customer’s specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

siemens.com/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

support.industry.siemens.com/cs/sc

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for iOS and Android:

support.industry.siemens.com/cs/ww/en/sc/2067

4.2 Industry Mall

The Siemens Industry Mall is the platform on which the entire siemens Industry product portfolio is accessible. From the selection of products to the order and the delivery tracking, the Industry Mall enables the complete purchasing processing – directly and independently of time and location:
mall.industry.siemens.com

4.3 Links and literature

Table 4-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the entry page of the application example https://support.industry.siemens.com/cs/ww/en/view/109779336
\3\	Wireshark https://www.wireshark.org/download.html

4.4 Change documentation

Table 4-2

Version	Date	Change
V1.0	05/2020	First edition
V2.0	11/2020	Added the "select" function
V2.1	02/2021	Added some notes in the documentation.
V3.0	05/2022	Complete revision
V3.1	11/2022	Bug fixes. Revision of the FB "AnalyzeTokens" (token detection, ERROR token).