SIEMENS

# Connecting an S7-1500 / S7-1200 to a SQL database

TIA Portal V17 / S7-1500 / S7-1200 Microsoft 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 protocol that facilitates 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).

Using TDS, you can log in to a SQL server database and transmit SQL instructions. In this way it is possible to read data from the database or send them to the database for storage.

This application example demonstrates how a SIMATIC S7-1500 uses the Open User Communication blocks (TCON, TSEND, TRCV and TDISCON) to establish a connection to a Microsoft SQL server and exchange data with a database.

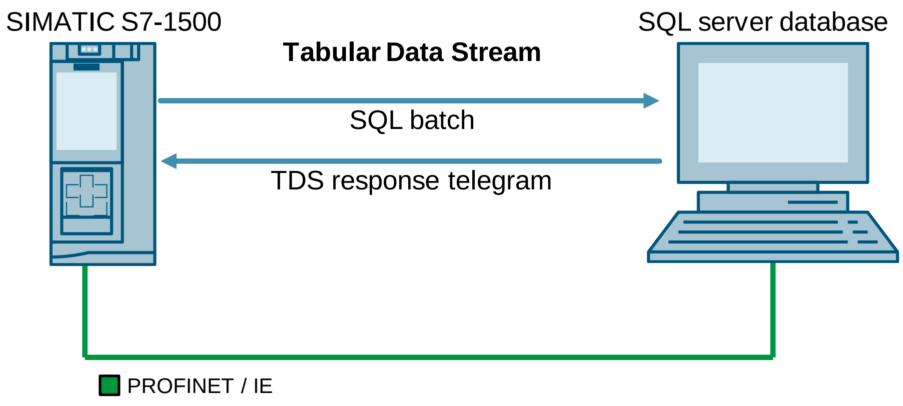
The following access operations to a Microsoft SQL server database are implemented in the application example:

- PRELOGIN: Send message 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 instructions:
  - SELECT
  - INSERT INTO
  - UPDATE
- Formulate and send SQL batch to execute stored procedures.
- Receive response from the SQL server to the executed SQL instruction.

### Overview of the application example

The Figure below provides an overview of the application example:

Figure 1-1



## 1.2 Principle of operation

### 1.2.1 Overview

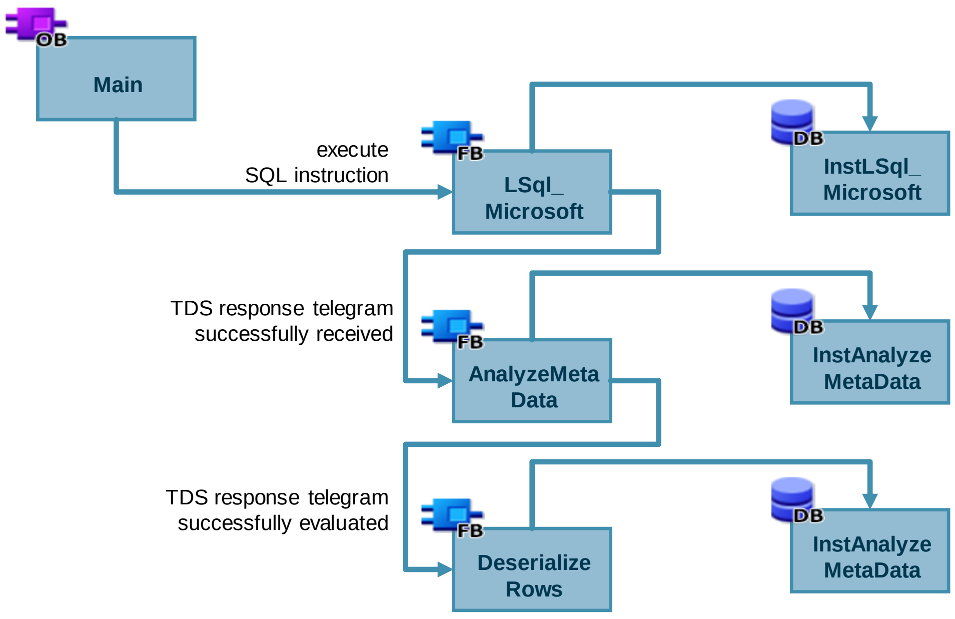
This application example contains the following function blocks (FBs):

Table 1-1

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

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

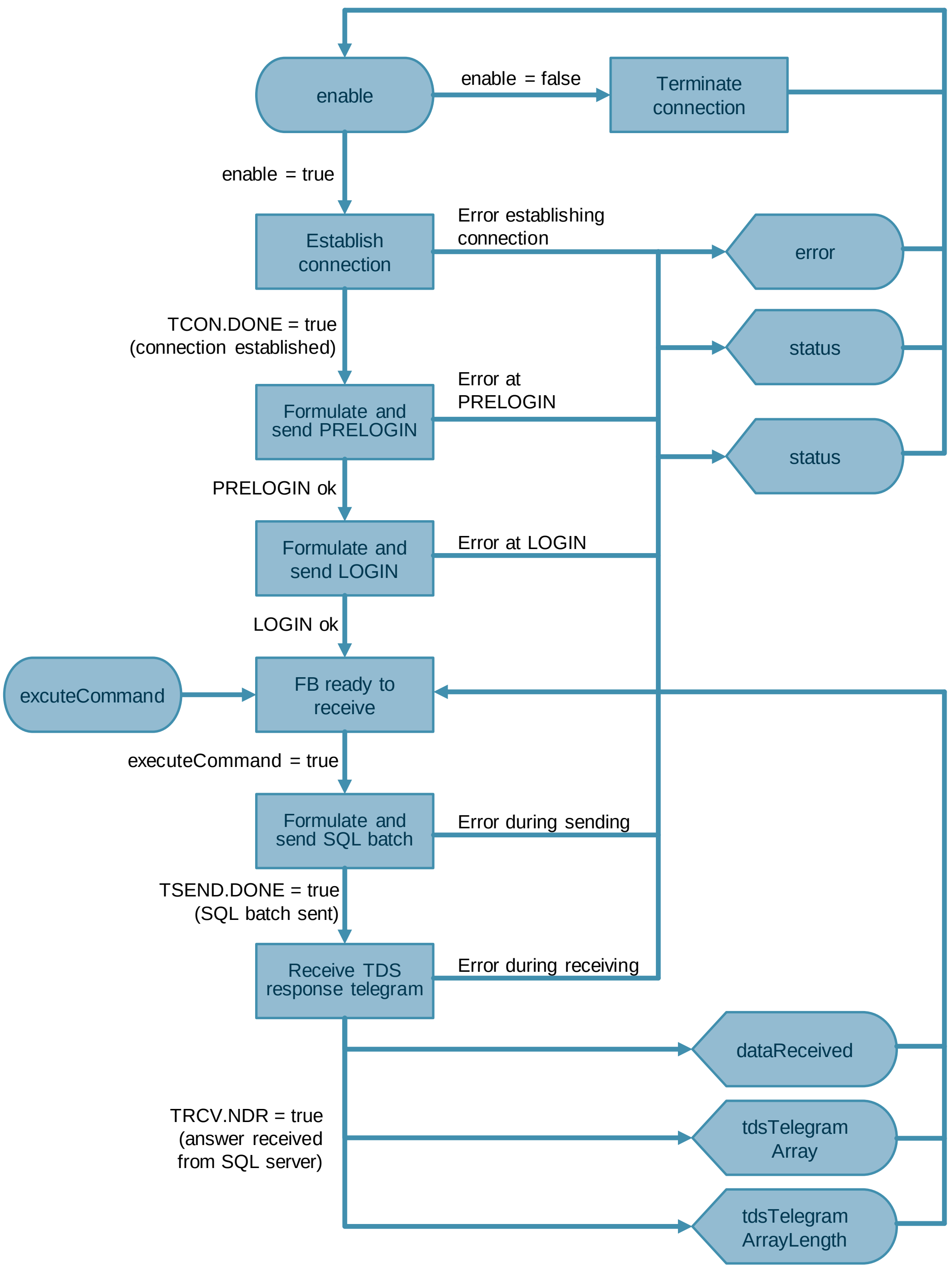
Figure 1-2



## Function block (FB) "LSql_Microsoft"

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

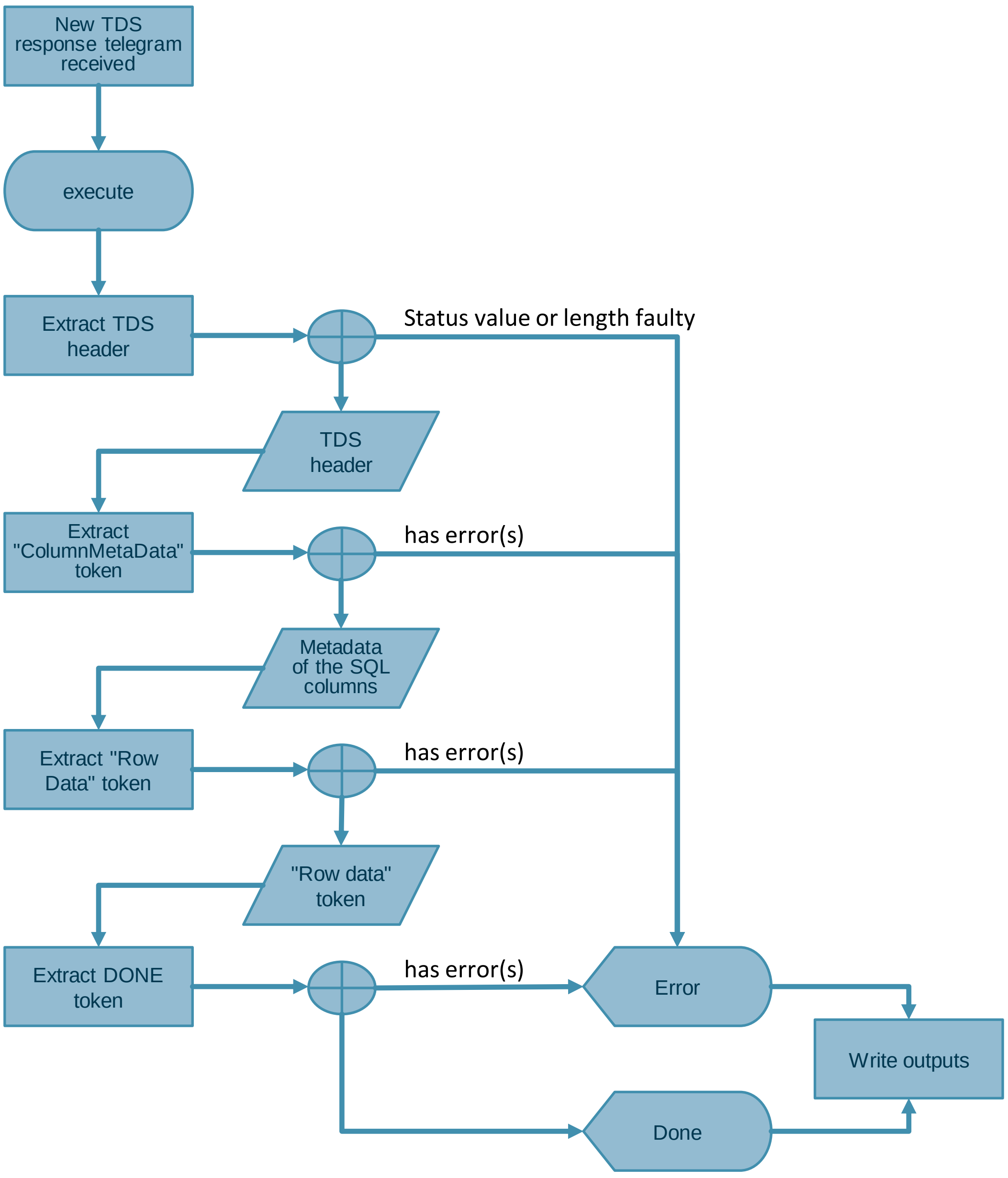
Figure 1-3



**Function block (FB) "AnalyzeMetaData"**

The Figure below shows the principle of operation and the structure of the FB "AnalyzeMetaData".

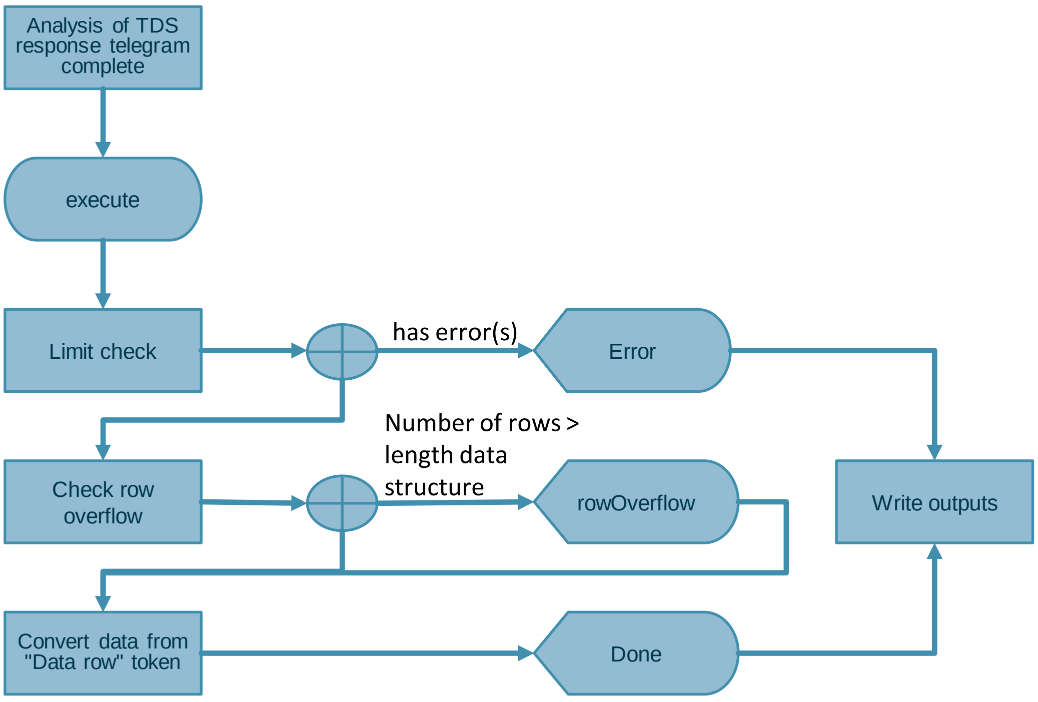
Figure 1-4



## Function block (FB) "DeserializeRows"

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

Figure 1-5


Analysis of TDS response telegram complete
execute
Limit check
has error(s)
Error
Number of rows > length data structure
Check row overflow
rowOverflow
Write outputs
Convert data from "Data row" token
Done

## 1.3 Components used

The following hardware and software components were used to create this application example:

Table 1-2

| Component | Quantity | Item number | Note |
|---|---|---|---|
| STEP 7 Professional V17 Update 2 | 1 | 6ES7822-1AA07-0YA7 | Engineering system |
| CPU 1513-1 PN | 1 | 6ES7513-1AL01-0AB0 | Alternatively, you can use another S7-1500 CPU or ET 200SP CPU with firmware V2.5 or later. |
| CPU 1214C | 1 | | Alternatively, you can use any other S7-1200 CPU with firmware V2.5 or later. |

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

This application example consists of the following components:

Table 1-3

| Component | File name | Note |
|---|---|---|
| Documentation | 109779336_SQL_DOC_en_V30.pdf | This document |
| Project | 109779336_SQL_CODE_V30.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_V30.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 FB "LSql_Microsoft" emulates the TDS protocol based on the "Open User Communication blocks". It facilitates the following actions:

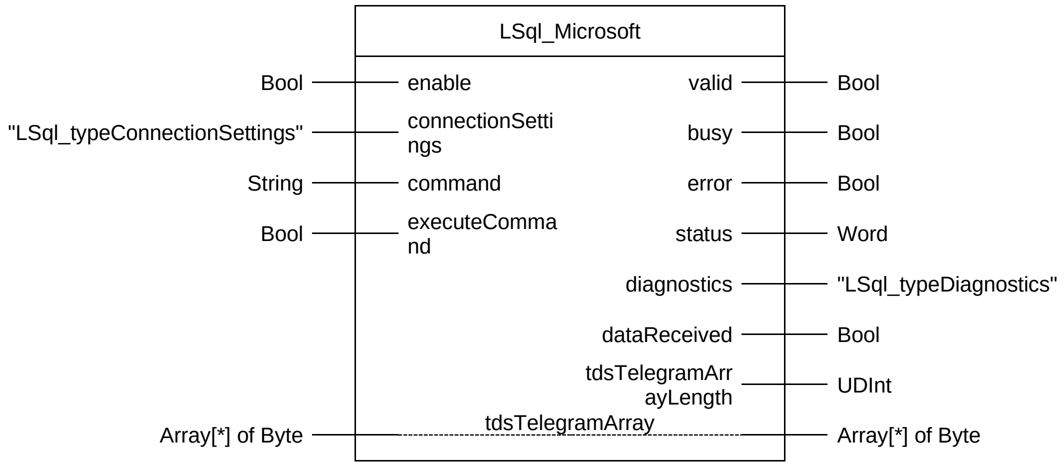
- Logging in to a Microsoft SQL server database (enable, connectionSettings)
- Transmitting SQL instructions (sqlCommand, executeCommand)
- Receiving read data (tokenRows, dataReceived)

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: LSql_Microsoft



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

Table 2-1

| Name | P Type | Data type | Description |
|---|---|---|---|
| enable | IN | Bool | Activates the function of the FB. |
| connectionSettings | IN | "LSql_typeConnectionSettings" | Parameters for connection setup and logging in to the database.<br>Detailed information about the PLC data type "LSql_typeConnectionSettings" can be found in Table 2-4. |
| command | IN | String | SQL command that is executed if executeCommand = TRUE. |
| executeCommand | IN | Bool | TRUE: SQL command will be executed once. |
| valid | OUT | Bool | TRUE: Valid values available at the outputs of the FB. |
| busy | OUT | Bool | TRUE: FB is not finished yet and new values at the outputs are to be 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<br>• 16#8000 - 16#FFFF: Error detection<br>Detailed information about the status messages and error messages can be found in chapter 2.5.1. |
| diagnostics | OUT | "LSql_typeDiagnostics" | Diagnostic information of the FB.<br>Detailed information about the PLC data type "LSql_typeDiagnostics" can be found in Table 2-11. |
| dataReceived | OUT | Bool | TRUE: New data available at the outputs. |
| tdsTelegramArrayLength | OUT | UDInt | Length (number of bytes) of the received TDS response telegram. |
| tdsTelegramArray | IN_OUT | Array[*] of Byte | Receive buffer for the TDS response telegram. |

### 2.1.2 FB "AnalyzeMetaData"

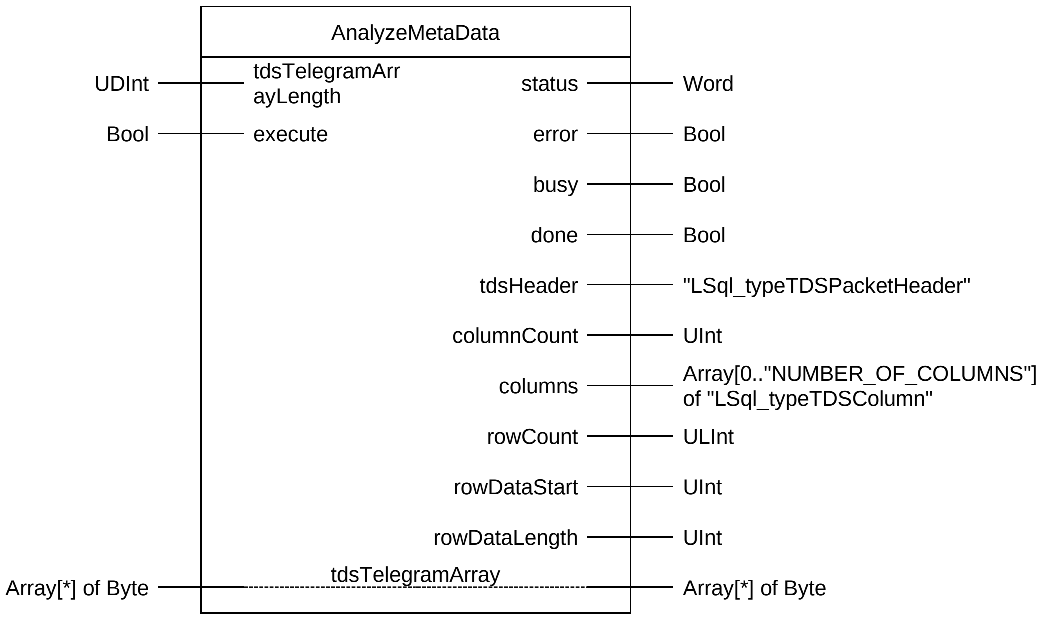
#### Functional description

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

#### Block interface

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

Figure 2-2



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

Table 2-2

| Name | P Type | Data type | Description |
|---|---|---|---|
| tdsTelegramArrayLength | IN | UDInt | Number of bytes received with TDS. |
| execute | IN | Bool | TRUE: FB execution initiated. |
| status | OUT | Word | Status display |
| error | OUT | Bool | Error display |
| busy | OUT | Bool | TRUE: FB is in process. |
| done | OUT | Bool | TRUE: The processing of the FB is complete and new values are available at the outputs of the FB. |
| tdsHeader | OUT | "LSql_typeTDSPacketHeader" | TDS header<br>Detailed information about the PLC data type "LSql_typeTDSPacketHeader" can be found in Table 2-7. |
| columnCount | OUT | UInt | Number of received columns. |

| Name | P Type | Data type | Description |
|---|---|---|---|
| columns | OUT | Array[0.."NUMBER_OF_COLUMNS"] of "LSql_typeTDSColumn" | Properties of the columns. Detailed information about the PLC data type "LSql_typeTDSColumn" can be found in Table 2-8. |
| rowCount | OUT | ULInt | Number of received rows. |
| rowDataStart | OUT | UInt | Address of the first byte of the "Row data" token stream. |
| rowDataLength | OUT | UInt | Length of the "Row Data" token stream. |
| tdsTelegramArray | IN_OUT | Array[*] of Byte | Receive buffer where the bytes received from the TDS response telegram are stored. |

### 2.1.3 FB "DeserializeRows"

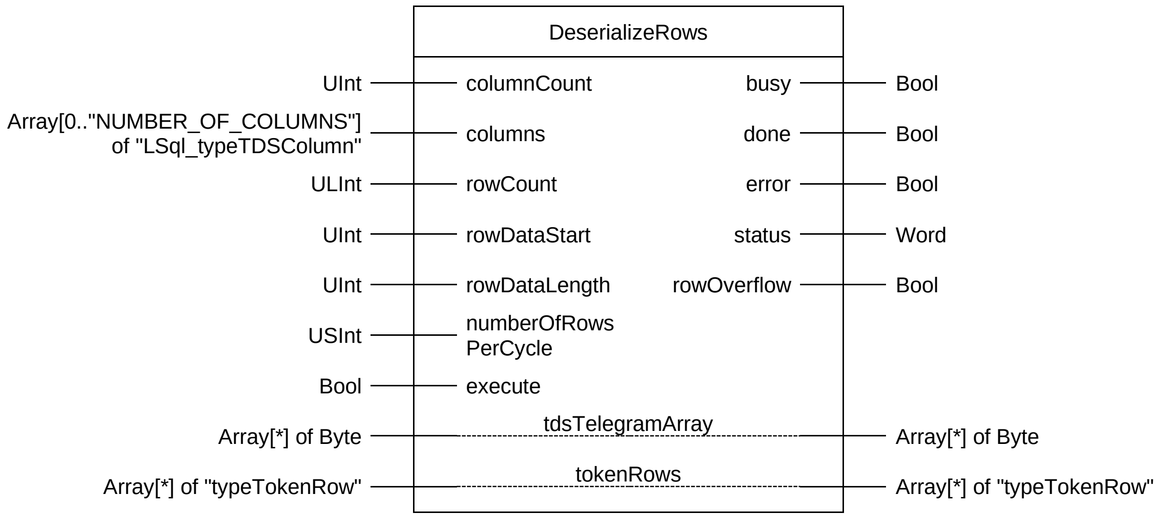
#### Functional description

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

#### 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 FB "DeserializeRows".

Table 2-3

| Name | P Type | Data type | Description |
|---|---|---|---|
| columnCount | IN | UInt | Number of received columns. |
| columns | IN | Array[0.."NUMBER_OF_COLUMNS"] of "LSql_typeTDSColumn" | Properties of the columns.<br>Detailed information about the PLC data type "LSql_typeTDSColumn" can be found in Table 2-8. |
| rowCount | IN | ULInt | Number of received rows. |
| rowDataStart | IN | UInt | Address of the first byte of the "Row data" token stream. |
| rowDataLength | IN | UInt | Length of the "Row Data" token stream. |
| numberOfRowsPerCycle | IN | USInt | Number of rows converted per cycle. |
| execute | IN | Bool | TRUE: FB execution initiated. |
| busy | OUT | Bool | TRUE: FB is in process. |
| done | OUT | Bool | TRUE: TRUE: The processing of the FB is complete and new values are available at the outputs of the FB. |
| error | OUT | Bool | Error display |
| status | OUT | Word | Status display |
| rowOverflow | OUT | Bool | TRUE = more rows were received from TDS than there are elements in the data structure "tokenRows". |
| tdsTelegramArray | IN_OUT | Array[*] of Byte | Receive buffer where the bytes received from the TDS response telegram are stored. |
| tokenRows | IN_OUT | Array[*] of "typeTokenRow" | User-specific data structure in which the output rows for the SQL command are stored.<br>The data structure depends on the user data.<br>For detailed information on the structure of the PLC data type, see Table 2-9. |

### 2.1.4 PLC data types

#### "LSql_typeConnectionSettings"

Table 2-4

| Parameters | Data type | Description |
|---|---|---|
| interfaceSettings | TCON_IP_v4 | IPv4 connection parameters. |
| loginInformation | "LSql_typeLoginInformation" | Login data for authentication between client and SQL server.<br>For detailed information on the structure of the PLC data type "LSql_typeLoginInformation", see Table 2-5. |

#### "LSql_typeLoginInformation"

Table 2-5

| Parameters | 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 with 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

| Parameters | Data type | Description |
|---|---|---|
| tdsHeader | "LSql_typeTDSPacketHeader" | TDS header<br>For detailed information on the structure of the PLC data type "LSql_typeTDSPacketHeader", refer to Table 2-7. |
| columns | Array[0.."NUMBER_OF_COLUMNS"] of "LSql_typeTDSColumn" | Data structure for the properties of the SQL columns.<br>For detailed information on the structure of the PLC data type "LSql_typeTDSColumn", see Table 2-8. |
| columnCount | UInt | Number of columns. |
| rowDataStart | UInt | Address of the first byte of the "Row data" token stream. |
| rowDataLength | UInt | Length of the "Row Data" token stream. |
| rowCount | ULInt | Number of received rows. |
| rowOverflow | Bool | TRUE = more rows were received from TDS than there are elements in the data structure "tokenRows". |
| tokenRows | Array[0.."NUMBER_OF_ROWS"] of "typeTokenRow" | User-specific data structure in which the output rows for the SQL command are stored.<br>The data structure depends on the user data.<br>For detailed information on the structure of the PLC data type "typeTokenRow", see Table 2-9. |

### "LSql_typeTDSPacketHeader"

Table 2-7

| Parameters | Data type | Description |
|---|---|---|
| type | Byte | Message type<br>4 = TDS response telegram from SQL server |
| status | Byte | Message status<br>0 = "normal" message<br>1 = End of message (EOM)<br>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 packet. |
| window | Byte | This parameter is currently unused. |

### "LSql_typeTDSColumn"

Table 2-8

| Parameters | Data type | Description |
|---|---|---|
| userType | ULInt | 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 the SQL columns.<br>For detailed information on the structure of the PLC data type "LSql_typeColumnFlag", see Table 2-10. |

### "typeTokenRow"

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

Table 2-9

| Parameters | Data type | Description |
|---|---|---|
| Amount | Int | Row value in column x, e. g. "Amount" |
| Color | String | Row value in column y, e. g. "Color" |
| Fruit | String | Row value in column z, e. g. "Fruit" |

**Note** Modify the name and data type of the parameters to fit the table in the SQL database.

### "LSql_typeColumnFlag"

Table 2-10

| Parameters | 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.<br>1: Column is readable and writable.<br>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

| Parameters | Data type | Description |
|---|---|---|
| status | Word | Status of the block or error identifier in case an error has occurred. |
| subfunctionStatus | Word | Status or returned value from called FBs, FCs and system blocks. |

## 2.2 Structure of the example database

The structure of the example database "SQLFromPLC" can be seen in the Table below.

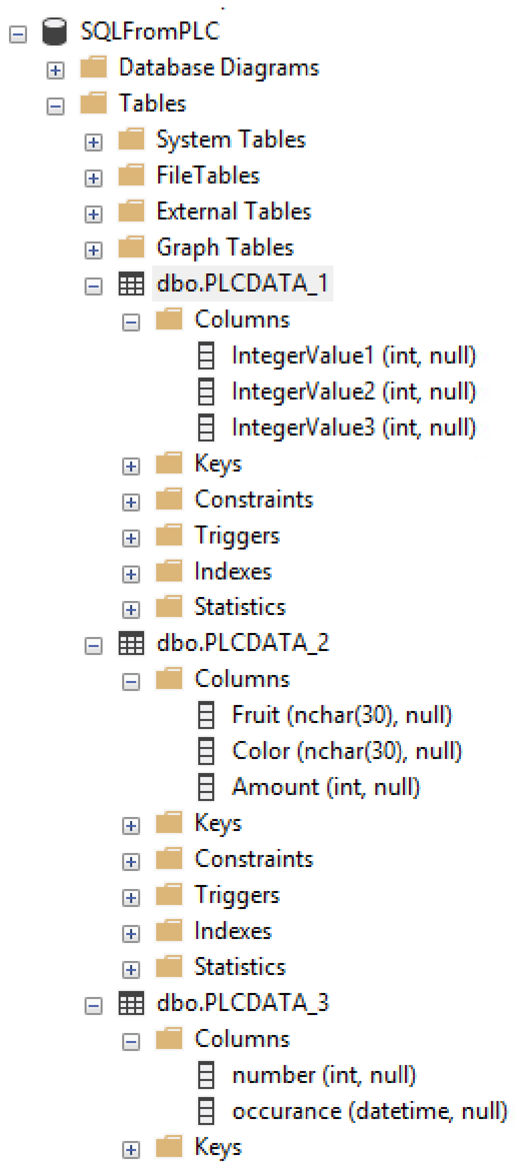
Table 2-12

| Table | Column name | Data type |
|---|---|---|
| PLCDATA_1 | IntegerValue1 | Int |
| | IntegerValue2 | Int |
| | IntegerValue3 | Int |
| PLCDATA_2 | Fruit | Nchar(30) |
| | Color | Nchar(30) |
| | Amount | Int |
| PLCDATA_3 | number | Int |
| | occurrence | datetime |

The structure has been chosen to demonstrate how numbers, strings and time stamps are saved.

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

Figure 2-4



```
SQLFromPLC
  Database Diagrams
  Tables
    System Tables
    FileTables
    External Tables
    Graph Tables
    dbo.PLCDATA_1
      Columns
        IntegerValue1 (int, null)
        IntegerValue2 (int, null)
        IntegerValue3 (int, null)
      Keys
      Constraints
      Triggers
      Indexes
      Statistics
    dbo.PLCDATA_2
      Columns
        Fruit (nchar(30), null)
        Color (nchar(30), null)
        Amount (int, null)
      Keys
      Constraints
      Triggers
      Indexes
      Statistics
    dbo.PLCDATA_3
      Columns
        number (int, null)
        occurance (datetime, null)
      Keys
```

## 2.3 Integration into the user project

### Requirements

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

- S7-1500 firmware V2.5 or later
- Microsoft SQL server is fully configured.
- S7-1500 and Microsoft SQL server are in the same subnet.

**Note** This block also works with an S7-1200 with firmware V4.4 or later.

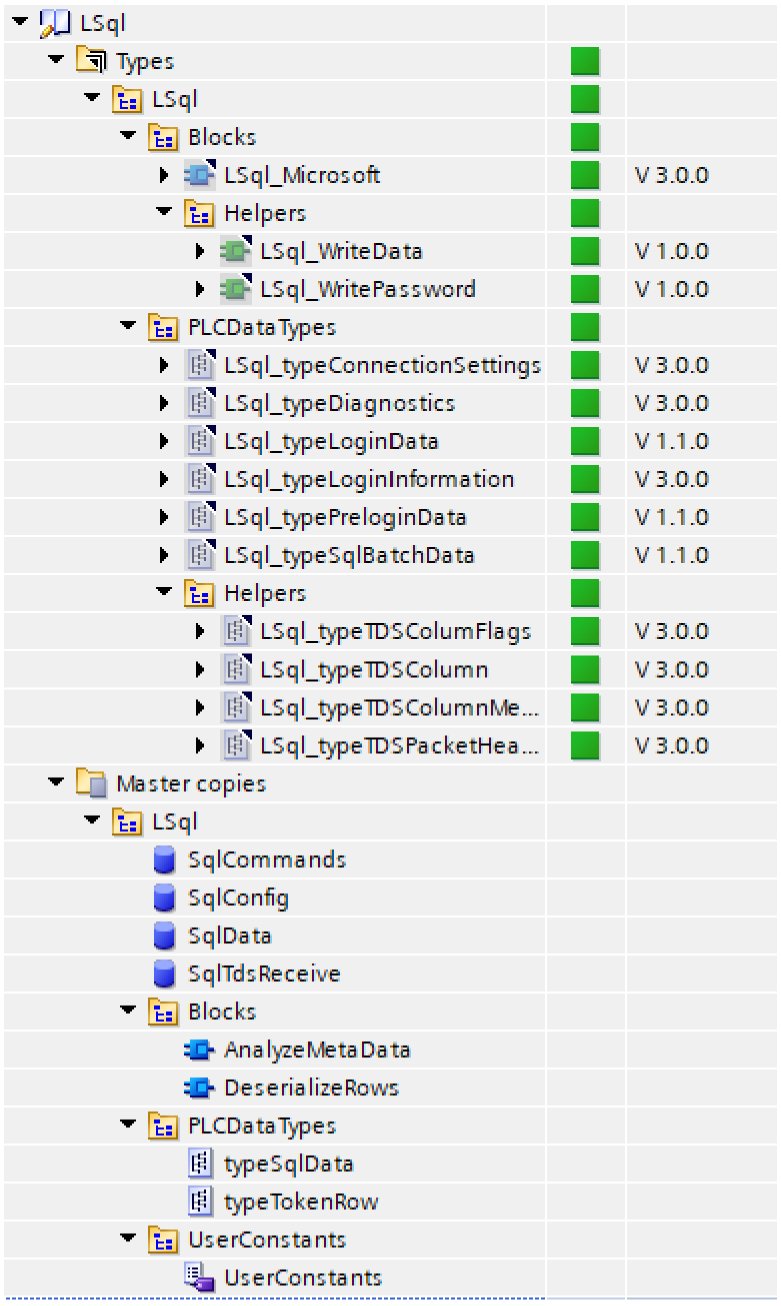
### Restrictions

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 command can be sent or received.
- Using Open User Communication with an S7-1200, a maximum of 8192 bytes per command can be sent or received.
- The block "LSql_Microsoft" may be called no more than once for each Microsoft SQL server connection.

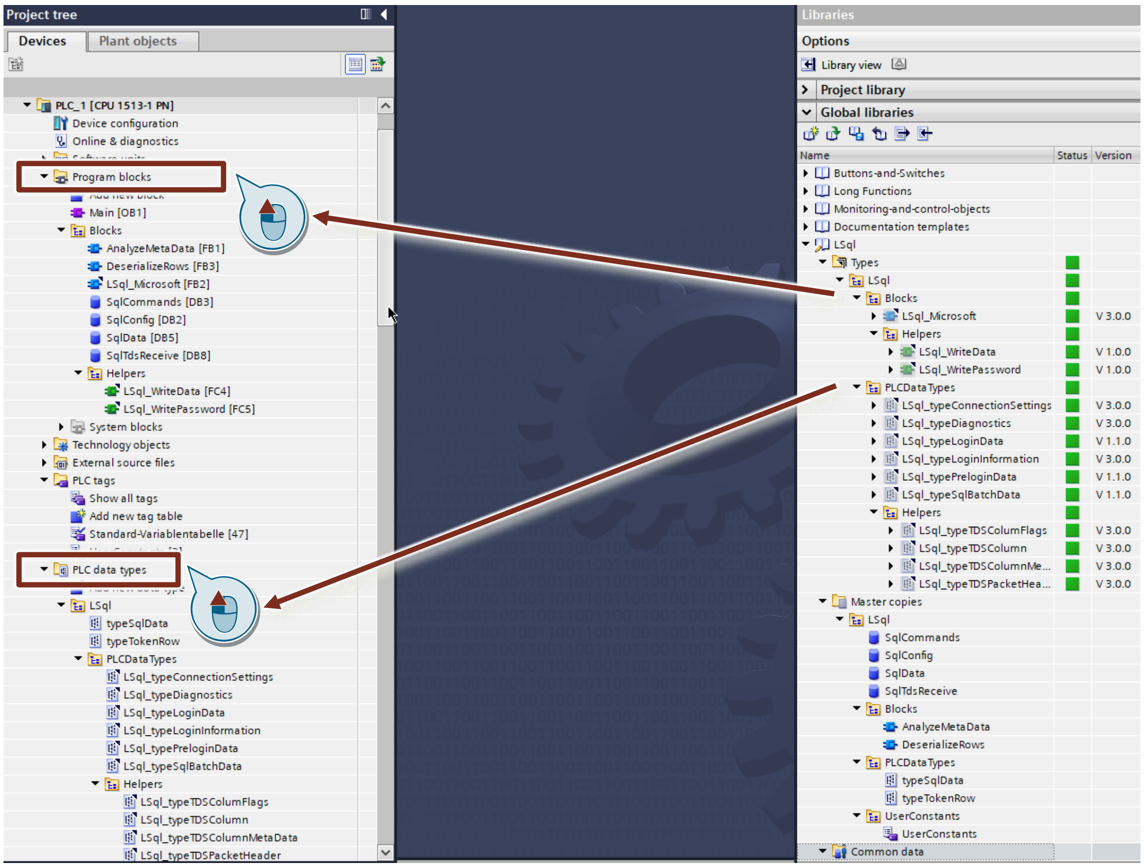
## Integrate library into user project

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



| | | |
|---|---|---|
| LSql | | |
| Types | 🟩 | |
| LSql | 🟩 | |
| Blocks | 🟩 | |
| LSql_Microsoft | 🟩 | V 3.0.0 |
| Helpers | 🟩 | |
| LSql_WriteData | 🟩 | V 1.0.0 |
| LSql_WritePassword | 🟩 | V 1.0.0 |
| PLCDataTypes | 🟩 | |
| LSql_typeConnectionSettings | 🟩 | V 3.0.0 |
| LSql_typeDiagnostics | 🟩 | V 3.0.0 |
| LSql_typeLoginData | 🟩 | V 1.1.0 |
| LSql_typeLoginInformation | 🟩 | V 3.0.0 |
| LSql_typePreloginData | 🟩 | V 1.1.0 |
| LSql_typeSqlBatchData | 🟩 | V 1.1.0 |
| Helpers | 🟩 | |
| LSql_typeTDSColumFlags | 🟩 | V 3.0.0 |
| LSql_typeTDSColumn | 🟩 | V 3.0.0 |
| LSql_typeTDSColumnMe... | 🟩 | V 3.0.0 |
| LSql_typeTDSPacketHea... | 🟩 | V 3.0.0 |
| Master copies | | |
| LSql | | |
| SqlCommands | | |
| SqlConfig | | |
| SqlData | | |
| SqlTdsReceive | | |
| Blocks | | |
| AnalyzeMetaData | | |
| DeserializeRows | | |
| PLCDataTypes | | |
| typeSqlData | | |
| typeTokenRow | | |
| UserConstants | | |
| UserConstants | | |

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"



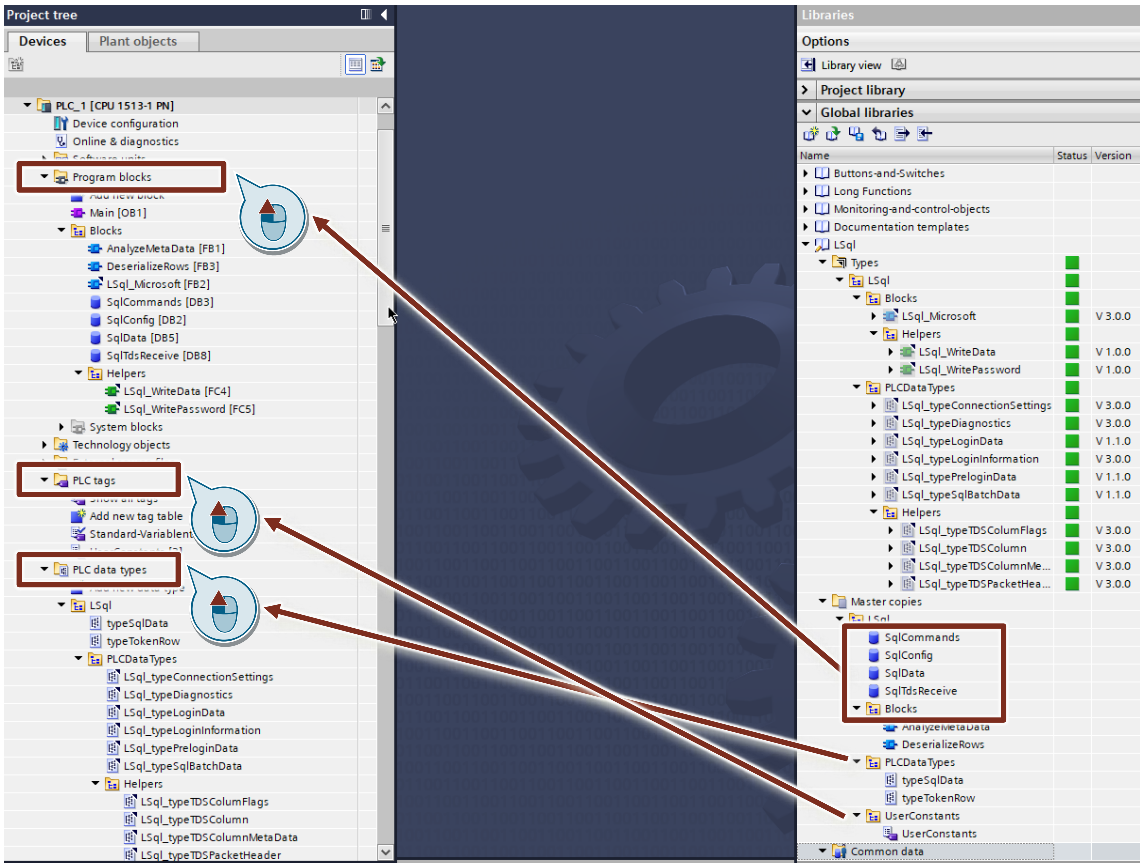
```
Project tree
Devices | Plant objects

PLC_1 [CPU 1513-1 PN]
  Device configuration
  Online & diagnostics
  Program blocks
    Main [OB1]
    Blocks
      AnalyzeMetaData [FB1]
      DeserializeRows [FB3]
      LSql_Microsoft [FB2]
      SqlCommands [DB3]
      SqlConfig [DB2]
      SqlData [DB5]
      SqlTdsReceive [DB8]
      Helpers
        LSql_WriteData [FC4]
        LSql_WritePassword [FC5]
    System blocks
  Technology objects
  External source files
  PLC tags
    Show all tags
    Add new tag table
    Standard-Variablentabelle [47]
  PLC data types
    LSql
      typeSqlData
      typeTokenRow
      PLCDataTypes
        LSql_typeConnectionSettings
        LSql_typeDiagnostics
        LSql_typeLoginData
        LSql_typeLoginInformation
        LSql_typePreloginData
        LSql_typeSqlBatchData
        Helpers
          LSql_typeTDSColumFlags
          LSql_typeTDSColumn
          LSql_typeTDSColumnMetaData
          LSql_typeTDSPacketHeader
```

```
Libraries
Options
Library view
> Project library
v Global libraries

Name                                         Status  Version
Buttons-and-Switches
Long Functions
Monitoring-and-control-objects
Documentation templates
LSql
  Types                                      🟩
    LSql                                     🟩
      Blocks                                 🟩
        LSql_Microsoft                       🟩      V 3.0.0
        Helpers                              🟩
          LSql_WriteData                     🟩      V 1.0.0
          LSql_WritePassword                 🟩      V 1.0.0
      PLCDataTypes                           🟩
        LSql_typeConnectionSettings          🟩      V 3.0.0
        LSql_typeDiagnostics                 🟩      V 3.0.0
        LSql_typeLoginData                   🟩      V 1.1.0
        LSql_typeLoginInformation            🟩      V 3.0.0
        LSql_typePreloginData                🟩      V 1.1.0
        LSql_typeSqlBatchData                🟩      V 1.1.0
        Helpers                              🟩
          LSql_typeTDSColumFlags             🟩      V 3.0.0
          LSql_typeTDSColumn                 🟩      V 3.0.0
          LSql_typeTDSColumnMe...            🟩      V 3.0.0
          LSql_typeTDSPacketHea...           🟩      V 3.0.0
  Master copies
    LSql
      SqlCommands
      SqlConfig
      SqlData
      SqlTdsReceive
      Blocks
        AnalyzeMetaData
        DeserializeRows
      PLCDataTypes
        typeSqlData
        typeTokenRow
      UserConstants
        UserConstants
  Common data
```

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"


Project tree
Devices | Plant objects

- PLC_1 [CPU 1513-1 PN]
  - Device configuration
  - Online & diagnostics
  - Program blocks
    - Main [OB1]
    - Blocks
      - AnalyzeMetaData [FB1]
      - DeserializeRows [FB3]
      - LSql_Microsoft [FB2]
      - SqlCommands [DB3]
      - SqlConfig [DB2]
      - SqlData [DB5]
      - SqlTdsReceive [DB8]
      - Helpers
        - LSql_WriteData [FC4]
        - LSql_WritePassword [FC5]
    - System blocks
  - Technology objects
  - PLC tags
    - Add new tag table
    - Standard-Variablent
  - PLC data types
    - LSql
      - typeSqlData
      - typeTokenRow
      - PLCDataTypes
        - LSql_typeConnectionSettings
        - LSql_typeDiagnostics
        - LSql_typeLoginData
        - LSql_typeLoginInformation
        - LSql_typePreloginData
        - LSql_typeSqlBatchData
        - Helpers
          - LSql_typeTDSColumFlags
          - LSql_typeTDSColumn
          - LSql_typeTDSColumnMetaData
          - LSql_typeTDSPacketHeader

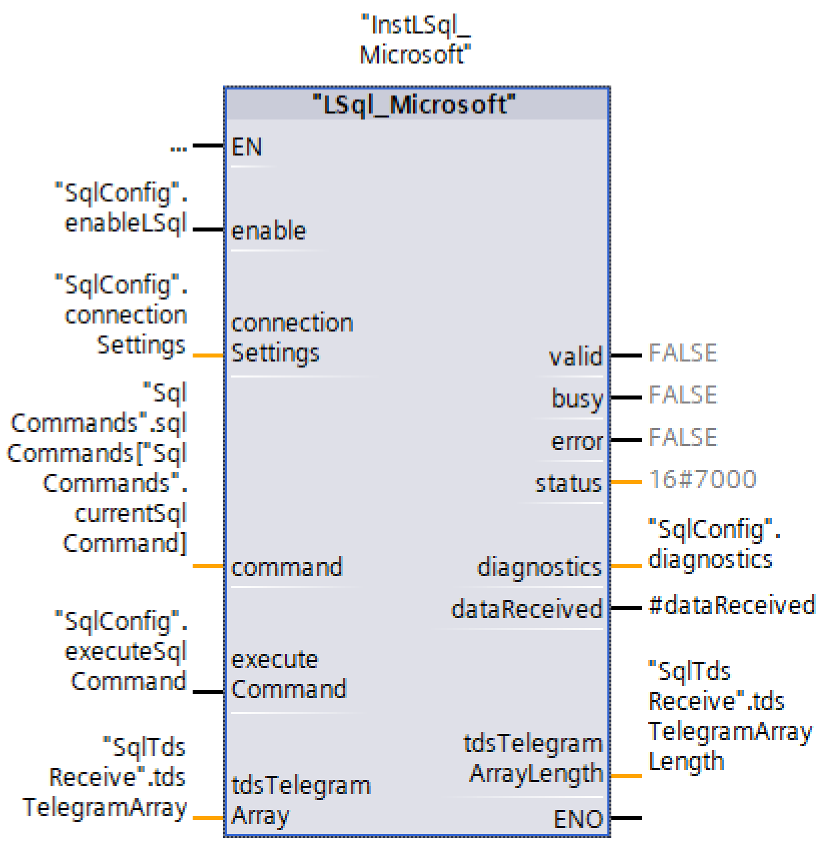
Libraries
Options
Library view
Project library
Global libraries

| Name | Status | Version |
|---|---|---|
| Buttons-and-Switches | | |
| Long Functions | | |
| Monitoring-and-control-objects | | |
| Documentation templates | | |
| LSql | | |
| Types | 🟩 | |
| LSql | 🟩 | |
| Blocks | 🟩 | |
| LSql_Microsoft | 🟩 | V 3.0.0 |
| Helpers | 🟩 | |
| LSql_WriteData | 🟩 | V 1.0.0 |
| LSql_WritePassword | 🟩 | V 1.0.0 |
| PLCDataTypes | 🟩 | |
| LSql_typeConnectionSettings | 🟩 | V 3.0.0 |
| LSql_typeDiagnostics | 🟩 | V 3.0.0 |
| LSql_typeLoginData | 🟩 | V 1.1.0 |
| LSql_typeLoginInformation | 🟩 | V 3.0.0 |
| LSql_typePreloginData | 🟩 | V 1.1.0 |
| LSql_typeSqlBatchData | 🟩 | V 1.1.0 |
| Helpers | 🟩 | |
| LSql_typeTDSColumFlags | 🟩 | V 3.0.0 |
| LSql_typeTDSColumn | 🟩 | V 3.0.0 |
| LSql_typeTDSColumnMe... | 🟩 | V 3.0.0 |
| LSql_typeTDSPacketHea... | 🟩 | V 3.0.0 |
| Master copies | | |
| LSql | | |
| SqlCommands | | |
| SqlConfig | | |
| SqlData | | |
| SqlTdsReceive | | |
| Blocks | | |
| AnalyzeMetaData | | |
| DeserializeRows | | |
| PLCDataTypes | | |
| typeSqlData | | |
| typeTokenRow | | |
| UserConstants | | |
| UserConstants | | |
| Common data | | |

## Interconnect the parameters of the FB "LSql_Microsoft"

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

Figure 2-5


"InstLSql_Microsoft"
"LSql_Microsoft"
... EN
"SqlConfig".enableLSql enable
"SqlConfig".connectionSettings connectionSettings
"SqlCommands".sqlCommands["SqlCommands".currentSqlCommand] command
"SqlConfig".executeSqlCommand executeCommand
"SqlTdsReceive".tdsTelegramArray tdsTelegramArray
valid FALSE
busy FALSE
error FALSE
status 16#7000
diagnostics "SqlConfig".diagnostics
dataReceived #dataReceived
tdsTelegramArrayLength "SqlTdsReceive".tdsTelegramArrayLength
ENO

**Note**

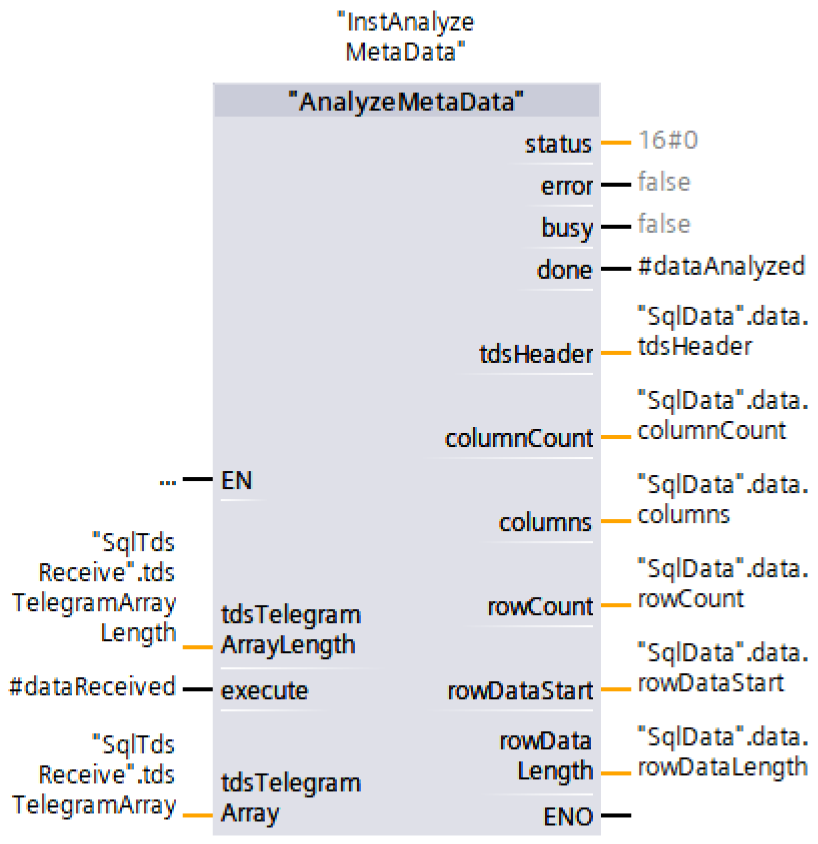
The wiring of the parameters described here is an essential requirement for operating the block.

On the SQL server side, it is necessary to allow the connection settings and login credentials to establish the connection. For detailed information on this topic, refer to chapter 3.2.

## Interconnect the parameters of the FB "AnalyzeMetaData"

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

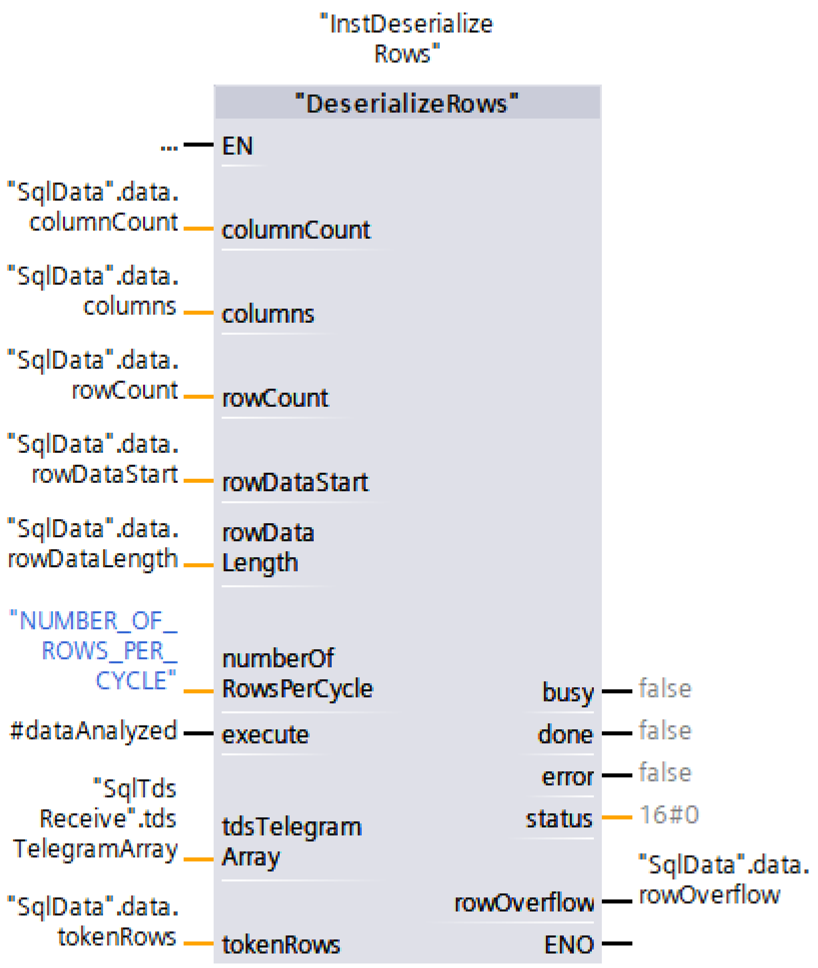
Figure 2-6


"InstAnalyze MetaData"
"AnalyzeMetaData"
... EN
"SqlTds Receive".tds TelegramArray Length — tdsTelegram ArrayLength
#dataReceived — execute
"SqlTds Receive".tds TelegramArray — tdsTelegram Array
status — 16#0
error — false
busy — false
done — #dataAnalyzed
tdsHeader — "SqlData".data. tdsHeader
columnCount — "SqlData".data. columnCount
columns — "SqlData".data. columns
rowCount — "SqlData".data. rowCount
rowDataStart — "SqlData".data. rowDataStart
rowData Length — "SqlData".data. rowDataLength
ENO

## Interconnect 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 (minimal interconnection).

Figure 2-7


"InstDeserialize Rows"
"DeserializeRows"
... EN
"SqlData".data.columnCount — columnCount
"SqlData".data.columns — columns
"SqlData".data.rowCount — rowCount
"SqlData".data.rowDataStart — rowDataStart
"SqlData".data.rowDataLength — rowDataLength
"NUMBER_OF_ROWS_PER_CYCLE" — numberOfRowsPerCycle
#dataAnalyzed — execute
"SqlTdsReceive".tdsTelegramArray — tdsTelegramArray
"SqlData".data.tokenRows — tokenRows
busy — false
done — false
error — false
status — 16#0
rowOverflow — "SqlData".data.rowOverflow
ENO

## 2.4 Operation of the FB "LSql_Microsoft"

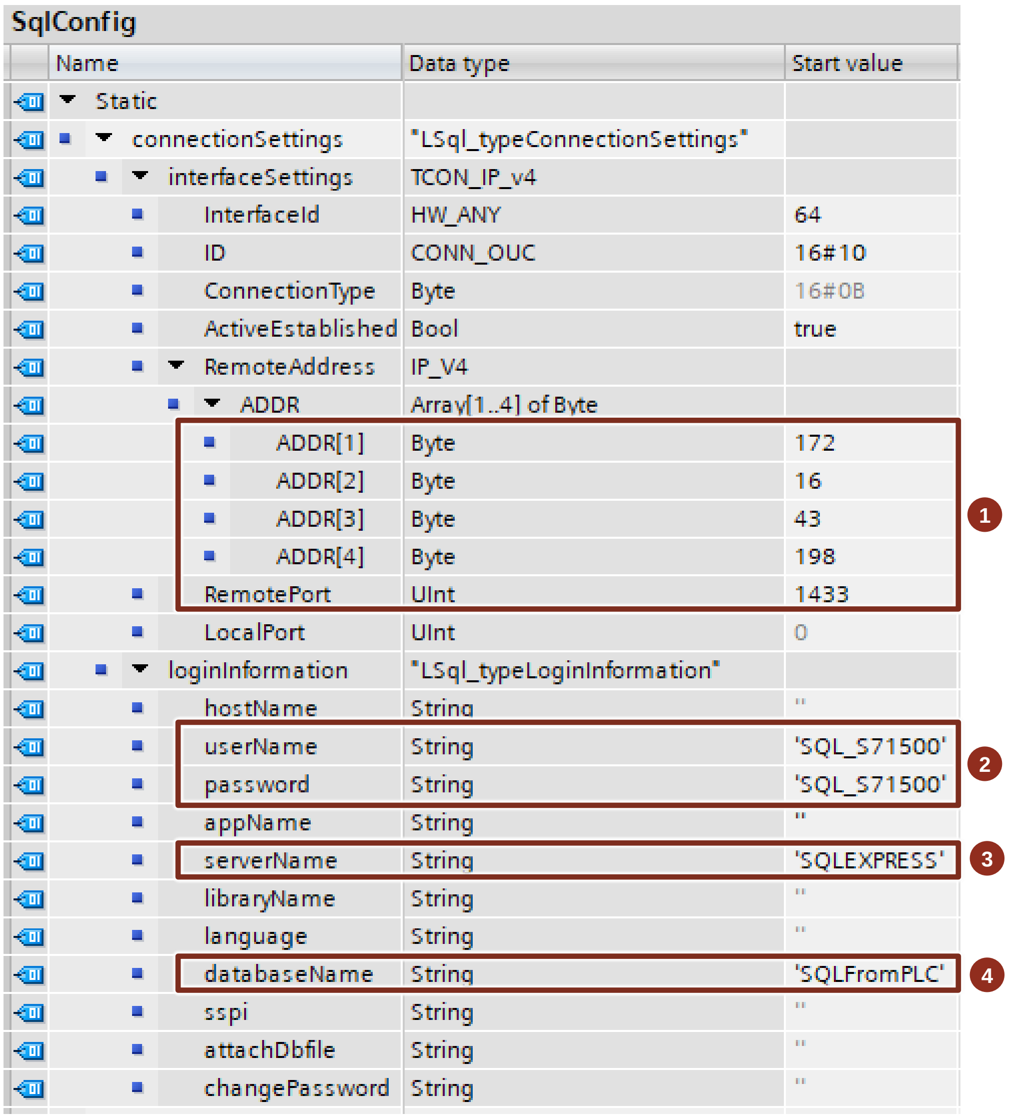
The "LSql_Microsoft" block is controlled via the "enable" and "executeCommand" inputs.

### 2.4.1 Establish and terminate connection

The "enable" input controls the process of establishing and terminating a connection to the SQL server. As long as SQL statements are to be transferred to the SQL server, "enable" must have the value. If "enable" is set to "FALSE" then the connection to the SQL server is terminated.

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 |
| 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' |
| password | String | 'SQL_S71500' |
| appName | String | '' |
| serverName | String | 'SQLEXPRESS' |
| libraryName | String | '' |
| language | String | '' |
| databaseName | String | 'SQLFromPLC' |
| sspi | String | '' |
| attachDbfile | String | '' |
| changePassword | String | '' |

Table 2-13

| | Parameters | Note |
|---|---|---|
| 1. | IP address and port of the SQL server | Default port for Microsoft SQL server is 1433. |
| 2. | SQL server login information | See chapter 3.2.1 |
| 3. | Name of SQL server | In this application example: SQLEXPRESS |
| 4. | Name of the database of the SQL server | An SQL server can contain multiple databases. Use this parameter to specify which database you wish to connect to. |

## 2.4.2 Transmit SQL instructions

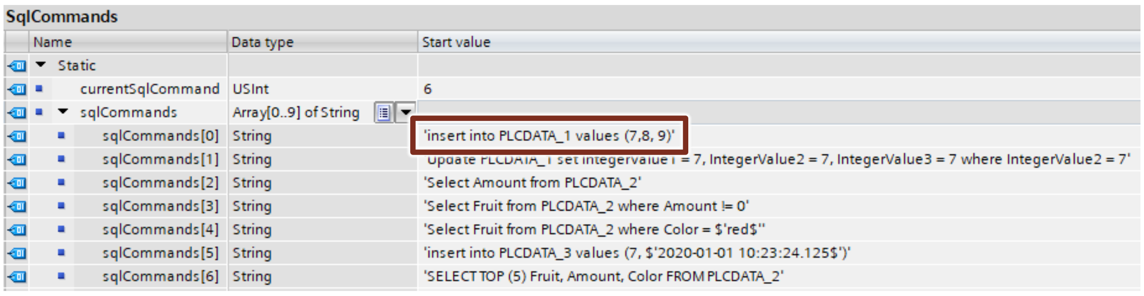
Formulate a SQL instruction and store it at the "command" input. Once the controller has established a connection to the SQL server, it is possible to transmit the SQL instruction to the SQL server with a rising edge at the "executeCommand" input.

**Note** The example only supports the standard ASCII encoding.

### 2.4.2.1 SQL instruction "insert into" – example integer values

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

Figure 2-9



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' |

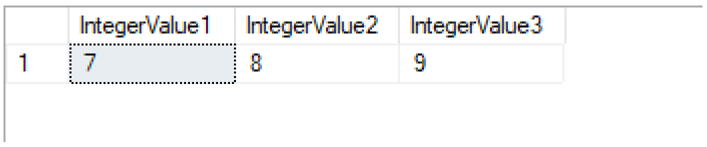
This SQL instruction inserts a new row into the database table "PLCDATA_1". Values (7,8,9) specifies the values that will be entered in the new row of the database table "PLCDATA_1".

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

A rising edge at the "executeCommand" input sends the SQL instruction to the database.

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

Figure 2-10

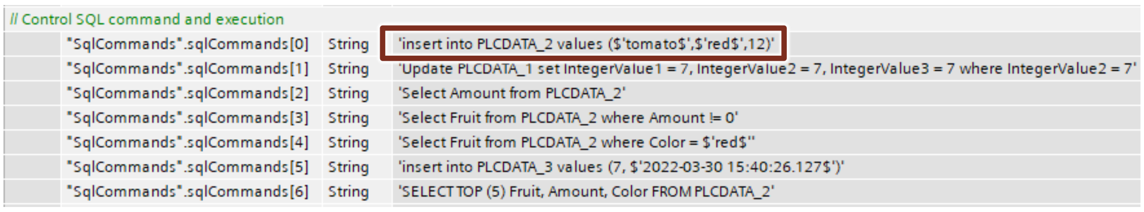


| | IntegerValue1 | IntegerValue2 | IntegerValue3 |
|---|---|---|---|
| 1 | 7 | 8 | 9 |

### 2.4.2.2 SQL instruction "insert into" – example strings

The following Figure shows an "insert into" SQL instruction for adding a new row containing strings into a database table.

Figure 2-11



// 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' |

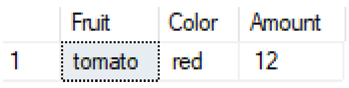
This SQL instruction inserts a new row into the database table "PLCDATA_2". Values ($'tomato$',$'red$',12) specifies the values that will be entered in the new row of the database table "PLCDATA_2".

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

A rising edge at the "executeCommand" input sends the SQL instruction to the database.

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

Figure 2-12

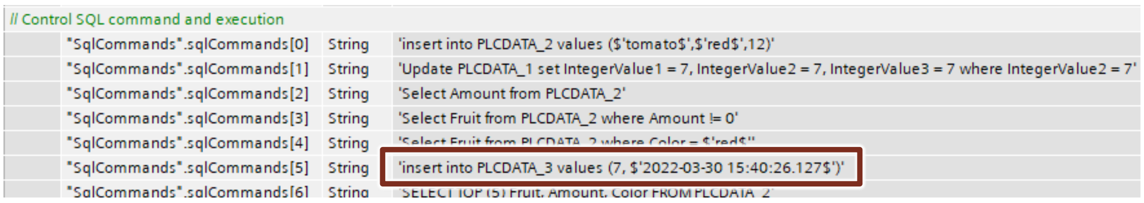


| | Fruit | Color | Amount |
|---|---|---|---|
| 1 | tomato | red | 12 |

### 2.4.2.3 SQL instruction "insert into" – example time stamps

The following Figure shows an "insert into" SQL instruction for adding a new row containing time stamps into a database table.

Figure 2-13



// 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' |

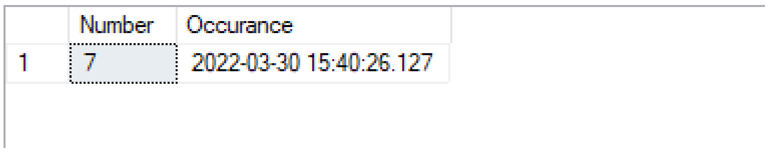
This SQL instruction inserts a new row into the database table "PLCDATA_3". Values (7,$'2022-03-30 15:40:26.125$') specifies the values that will be entered in the new row of the database table "PLCDATA_3".

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

A rising edge at the "executeCommand" input sends the SQL instruction to the database.

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

Figure 2-14

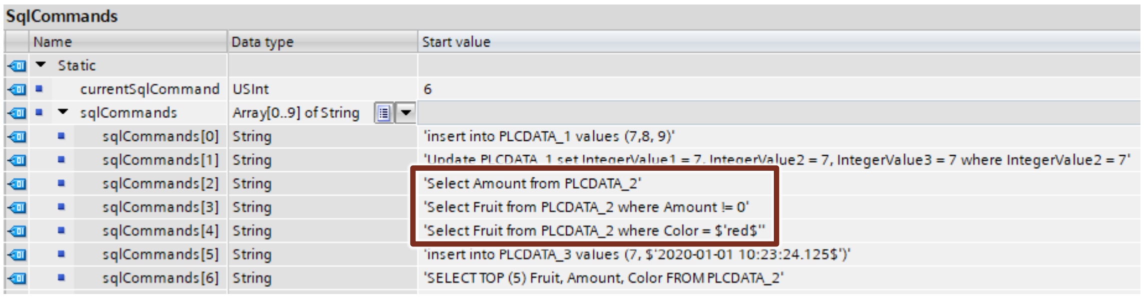


| | Number | Occurance |
|---|---|---|
| 1 | 7 | 2022-03-30 15:40:26.127 |

### 2.4.2.4 The "select" SCL instruction

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

Figure 2-15



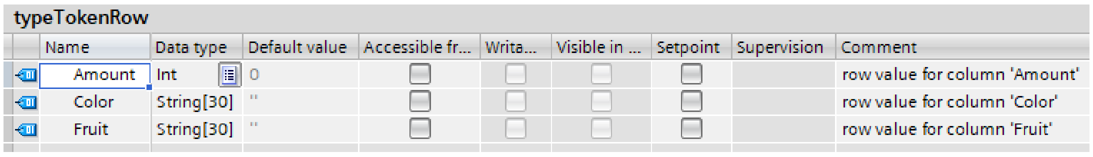
**SqlCommands**

| 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 instructions read values from the column "Fruit" in the database table "PLCDATA_2".

Below, we demonstrate how the SQL instruction "select" works and which adjustments you will need to make for your query.

1. Modify the PLC data type "typeTokenRow" to match the database table from which the data will be read with the "select" SQL instruction.
   In this application example, the PLC data type "typeTokenRow" has been adjusted to match the "PLCDATA_2" database table.

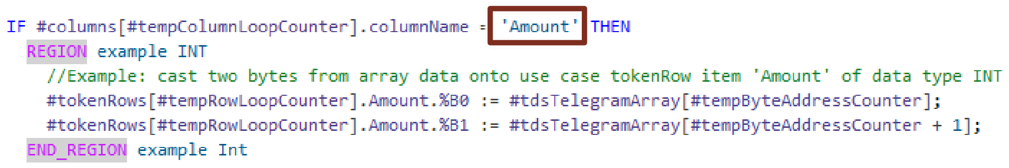


**typeTokenRow**

| Name | Data type | Default value | Accessible fr... | Writa... | Visible in ... | Setpoint | Supervision | Comment |
|---|---|---|---|---|---|---|---|---|
| Amount | Int | 0 | ☐ | ☐ | ☐ | ☐ | | row value for column 'Amount' |
| Color | String[30] | '' | ☐ | ☐ | ☐ | ☐ | | row value for column 'Color' |
| Fruit | String[30] | '' | ☐ | ☐ | ☐ | ☐ | | row value for column 'Fruit' |

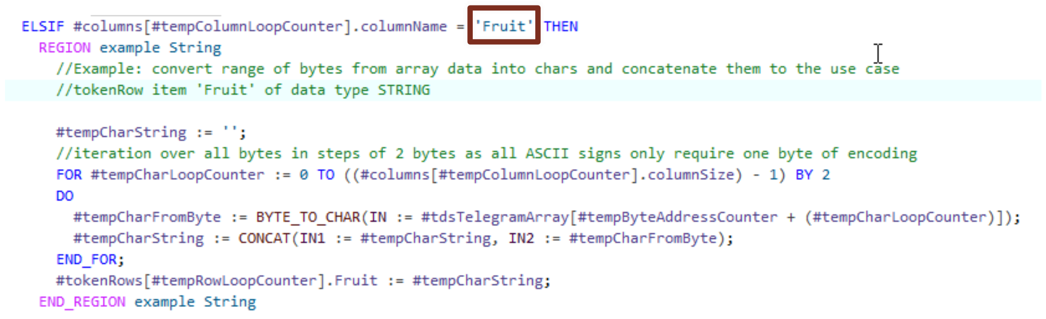
2. In the FB "DeserializeRows" you will find examples for reading out strings, integer values and Boolean values.
   In the query, modify the column names to match the column names in the database table that you will read.
   - The following Figure shows the process of reading integer values from the "Amount" column.

     Figure 2-16



```
IF #columns[#tempColumnLoopCounter].columnName = 'Amount' THEN
  REGION example INT
    //Example: cast two bytes from array data onto use case tokenRow item 'Amount' of data type INT
    #tokenRows[#tempRowLoopCounter].Amount.%B0 := #tdsTelegramArray[#tempByteAddressCounter];
    #tokenRows[#tempRowLoopCounter].Amount.%B1 := #tdsTelegramArray[#tempByteAddressCounter + 1];
  END_REGION example Int
```

   - The Figure below shows the process of reading strings from the "Fruit" column.

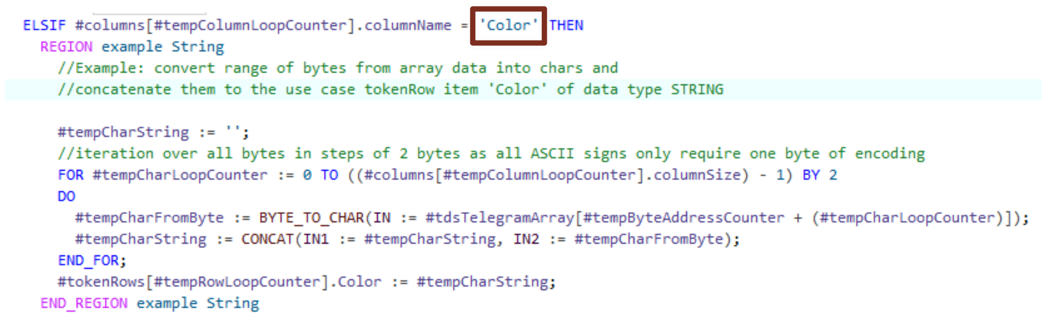


```
ELSIF #columns[#tempColumnLoopCounter].columnName = 'Fruit' THEN
  REGION example String
    //Example: convert range of bytes from array data into chars and concatenate them to the use case
    //tokenRow item 'Fruit' of data type STRING

    #tempCharString := '';
    //iteration over all bytes in steps of 2 bytes as all ASCII signs only require one byte of encoding
    FOR #tempCharLoopCounter := 0 TO ((#columns[#tempColumnLoopCounter].columnSize) - 1) BY 2
    DO
      #tempCharFromByte := BYTE_TO_CHAR(IN := #tdsTelegramArray[#tempByteAddressCounter + (#tempCharLoopCounter)]);
      #tempCharString := CONCAT(IN1 := #tempCharString, IN2 := #tempCharFromByte);
    END_FOR;
    #tokenRows[#tempRowLoopCounter].Fruit := #tempCharString;
  END_REGION example String
```

- The Figure below shows the process of reading strings from the "Color" column.

  Figure 2-17



```
ELSIF #columns[#tempColumnLoopCounter].columnName = 'Color' THEN
  REGION example String
    //Example: convert range of bytes from array data into chars and
    //concatenate them to the use case tokenRow item 'Color' of data type STRING

    #tempCharString := '';
    //iteration over all bytes in steps of 2 bytes as all ASCII signs only require one byte of encoding
    FOR #tempCharLoopCounter := 0 TO ((#columns[#tempColumnLoopCounter].columnSize) - 1) BY 2
    DO
      #tempCharFromByte := BYTE_TO_CHAR(IN := #tdsTelegramArray[#tempByteAddressCounter + (#tempCharLoopCounter)]);
      #tempCharString := CONCAT(IN1 := #tempCharString, IN2 := #tempCharFromByte);
    END_FOR;
    #tokenRows[#tempRowLoopCounter].Color := #tempCharString;
  END_REGION example String
```

3. Insert any additional queries as needed.

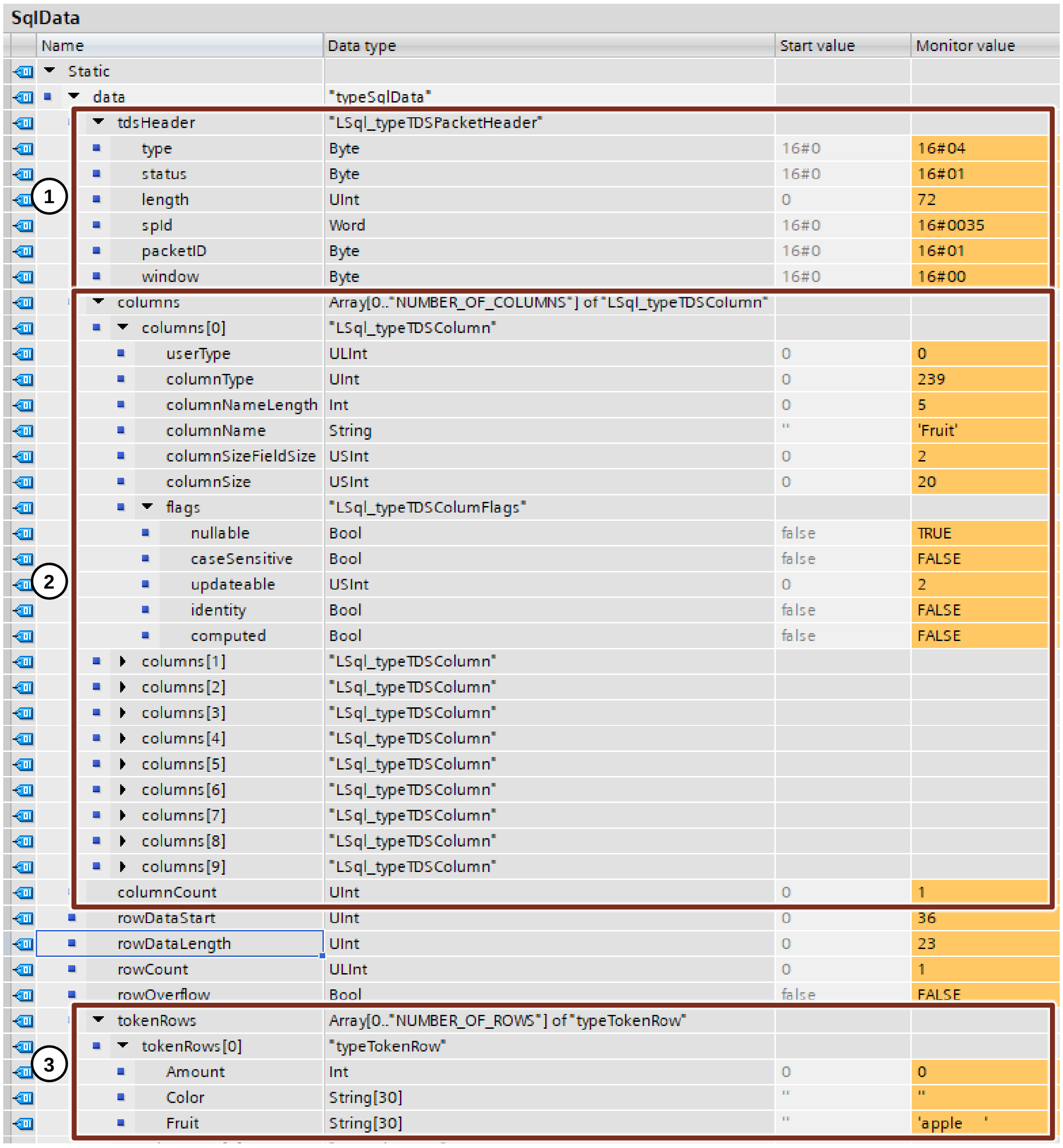
**Result**

The data you have read are now contained in the TDS response telegram. The TDS response telegram is stored in the "SqlTdsReceive" data block (DB) in the data structure "tdsTelegramArray".

The FBs "AnalyzeMetaData" and "DeserializeRows" prepare the TDS response telegram in such a way that the data can be read by the user.

The prepared data is stored in the DB "Sql Data" in the "typeSqlData" PLC data type structure.

Figure 2-18



SqlData

| Name | Data type | Start value | Monitor value |
|---|---|---|---|
| Static | | | |
| data | "typeSqlData" | | |
| tdsHeader | "LSql_typeTDSPacketHeader" | | |
| type | Byte | 16#0 | 16#04 |
| status | Byte | 16#0 | 16#01 |
| length | UInt | 0 | 72 |
| spId | Word | 16#0 | 16#0035 |
| packetID | Byte | 16#0 | 16#01 |
| window | Byte | 16#0 | 16#00 |
| columns | Array[0.."NUMBER_OF_COLUMNS"] of "LSql_typeTDSColumn" | | |
| columns[0] | "LSql_typeTDSColumn" | | |
| userType | ULInt | 0 | 0 |
| columnType | UInt | 0 | 239 |
| columnNameLength | Int | 0 | 5 |
| columnName | String | '' | 'Fruit' |
| columnSizeFieldSize | USInt | 0 | 2 |
| columnSize | USInt | 0 | 20 |
| flags | "LSql_typeTDSColumFlags" | | |
| nullable | Bool | false | TRUE |
| caseSensitive | Bool | false | FALSE |
| updateable | USInt | 0 | 2 |
| identity | Bool | false | FALSE |
| computed | Bool | false | FALSE |
| columns[1] | "LSql_typeTDSColumn" | | |
| columns[2] | "LSql_typeTDSColumn" | | |
| columns[3] | "LSql_typeTDSColumn" | | |
| columns[4] | "LSql_typeTDSColumn" | | |
| columns[5] | "LSql_typeTDSColumn" | | |
| columns[6] | "LSql_typeTDSColumn" | | |
| columns[7] | "LSql_typeTDSColumn" | | |
| columns[8] | "LSql_typeTDSColumn" | | |
| columns[9] | "LSql_typeTDSColumn" | | |
| columnCount | UInt | 0 | 1 |
| rowDataStart | UInt | 0 | 36 |
| rowDataLength | UInt | 0 | 23 |
| rowCount | ULInt | 0 | 1 |
| rowOverflow | Bool | false | FALSE |
| tokenRows | Array[0.."NUMBER_OF_ROWS"] of "typeTokenRow" | | |
| tokenRows[0] | "typeTokenRow" | | |
| Amount | Int | 0 | 0 |
| Color | String[30] | '' | '' |
| Fruit | String[30] | '' | 'apple ' |

The Table below contains a description of the data that were read.

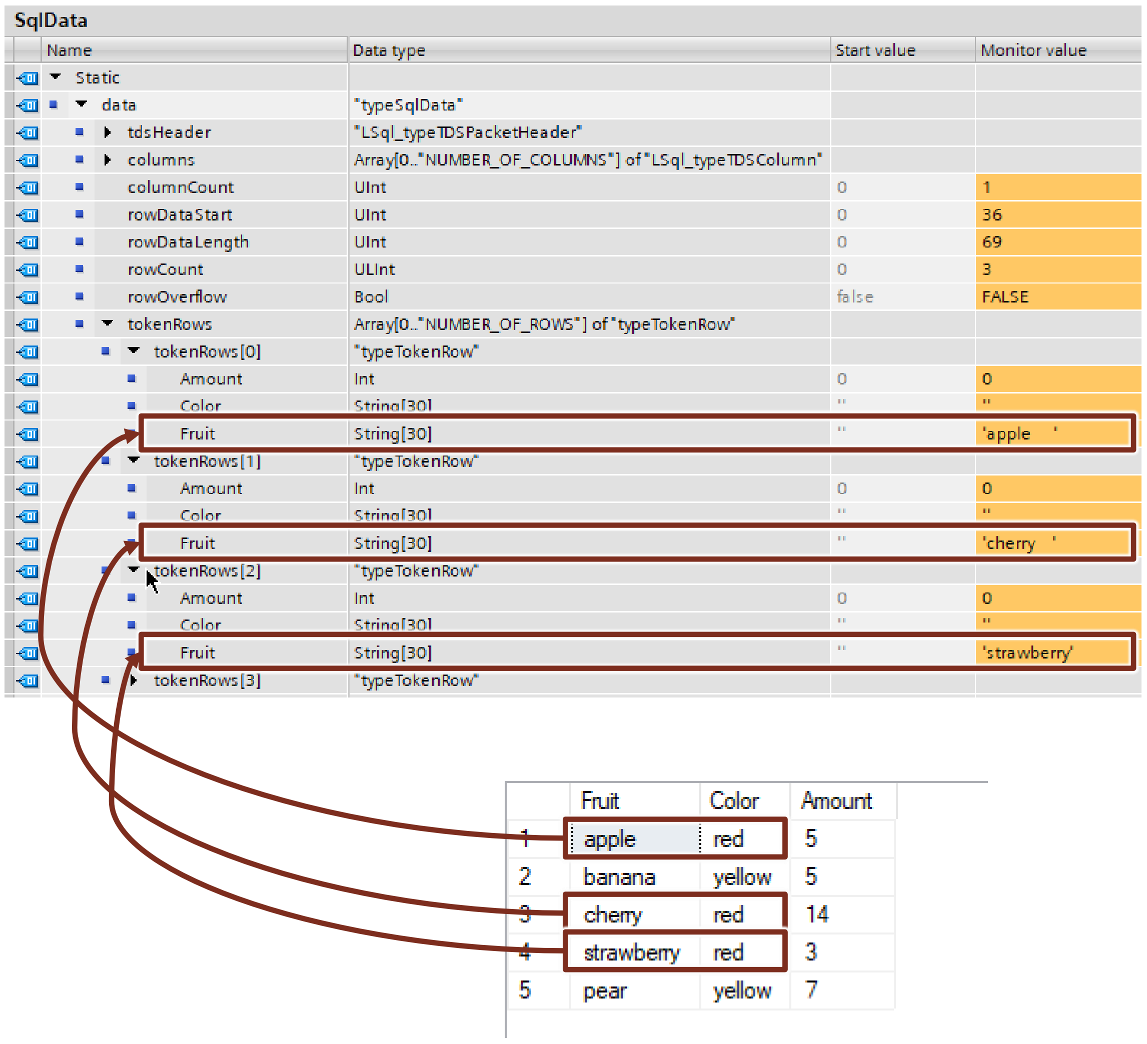
Table 2-14

| No. | Description of read data |
|---|---|
| 1. | TDS header |
| 2. | Metadata of the SQL columns<br>• Number of columns<br>• 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 "tokenRows[x]" array.

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

Figure 2-19



SqlData

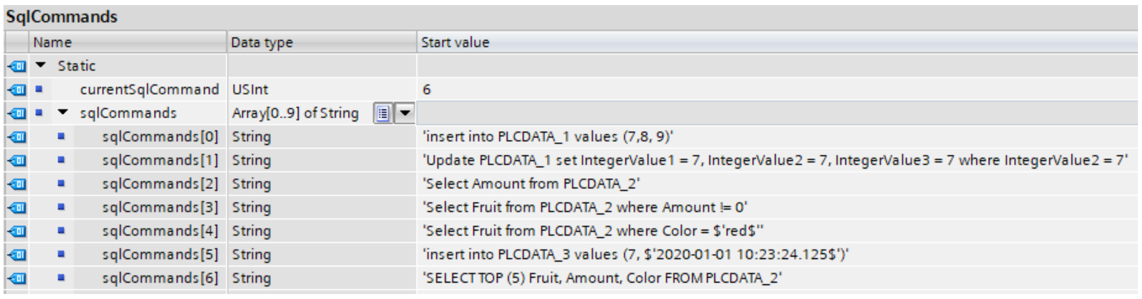
| Name | Data type | Start value | Monitor value |
|---|---|---|---|
| Static | | | |
| data | "typeSqlData" | | |
| tdsHeader | "LSql_typeTDSPacketHeader" | | |
| columns | Array[0.."NUMBER_OF_COLUMNS"] of "LSql_typeTDSColumn" | | |
| columnCount | UInt | 0 | 1 |
| rowDataStart | UInt | 0 | 36 |
| rowDataLength | UInt | 0 | 69 |
| rowCount | ULInt | 0 | 3 |
| rowOverflow | Bool | false | FALSE |
| tokenRows | Array[0.."NUMBER_OF_ROWS"] of "typeTokenRow" | | |
| tokenRows[0] | "typeTokenRow" | | |
| Amount | Int | 0 | 0 |
| Color | String[30] | '' | '' |
| Fruit | String[30] | '' | 'apple ' |
| tokenRows[1] | "typeTokenRow" | | |
| Amount | Int | 0 | 0 |
| Color | String[30] | '' | '' |
| Fruit | String[30] | '' | 'cherry ' |
| tokenRows[2] | "typeTokenRow" | | |
| Amount | Int | 0 | 0 |
| Color | String[30] | '' | '' |
| Fruit | String[30] | '' | 'strawberry' |
| tokenRows[3] | "typeTokenRow" | | |

| | Fruit | Color | Amount |
|---|---|---|---|
| 1 | apple | red | 5 |
| 2 | banana | yellow | 5 |
| 3 | cherry | red | 14 |
| 4 | strawberry | red | 3 |
| 5 | pear | yellow | 7 |

### 2.4.2.5 Additional SQL instructions

You can find more examples of SQL instructions in the "SqlCommands" data block as seen in the Figure below.

Figure 2-20



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** Chapter 3.5 provides information on calling stored procedures.

## 2.5 Troubleshooting

If one of the FBs has an error, the output parameters "error" and "status" must be investigated. 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.

### 2.5.1 Status and error messages

#### FB "LSql_Microsoft"

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

Table 2-15

| 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 new incoming 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#8600 | Error due to an undefined state in the state machine. |
| ERR_TDISCONNET | 16#8601 | Error when calling TDISCON. |
| ERR_CONNECT | 16#8602 | Error when calling TCON. |
| ERR_PRELOGIN | 16#8603 | Error executing the prelogin. |
| ERR_PRELOGIN_DATA | 16#8607 | Error converting data for the prelogin |
| ERR_LOGIN | 16#8604 | Error executing the login. |
| ERR_LOGIN_DATA | 16#8608 | Error converting the data for the login. |
| ERR_TRCV | 16#8605 | Error when calling TRCV. |
| ERR_SQLBATCH_DATA | 16#8609 | Error converting data for the SQL instruction. |
| ERR_SQLBATCH_SEND | 16#8610 | Error when sending SQL data. |

## FB "AnalyzeMetaData"

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

Table 2-16

| Error message / name of the constant in the FB | Value | Description |
|---|---|---|
| ERR_COLUMNMETADATA_WRONGTOKENTYPE | 16#8601 | Wrong type for the "ColumnMetaData" token stream. |
| ERR_COLUMNMETADATA_INCORRECTLENGTH | 16#8602 | Wrong length for the "ColumnMetaData" token stream. |
| ERR_COLUMNS_SIZENOTDEFINED | 16#8603 | At least one column variable 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_WRONGSTATUS | 16#8606 | The DONE token has the wrong status. |
| ERR_TOKENDONE_WRONGTOKENTYPE | 16#8607 | The DONE token has the wrong token type. |
| ERR_HEADER_WRONGSTATUS | 16#8608 | The TDS header has the wrong status. |

## FB "DeserializeRows"

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

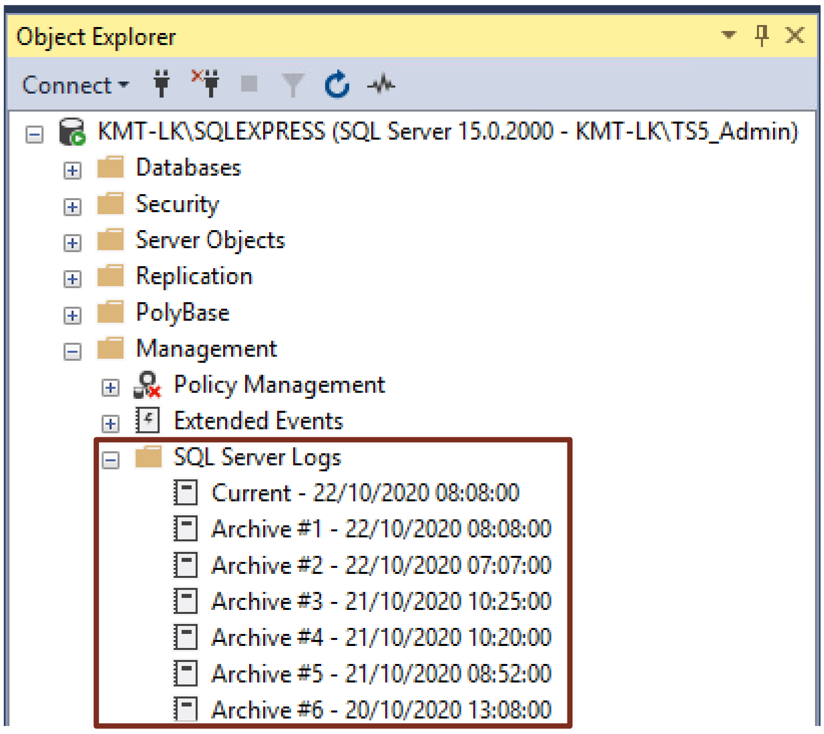
Table 2-17

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

## 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 2-21


Object Explorer
Connect
KMT-LK\SQLEXPRESS (SQL Server 15.0.2000 - KMT-LK\TS5_Admin)
Databases
Security
Server Objects
Replication
PolyBase
Management
Policy Management
Extended Events
SQL Server Logs
Current - 22/10/2020 08:08:00
Archive #1 - 22/10/2020 08:08:00
Archive #2 - 22/10/2020 07:07:00
Archive #3 - 21/10/2020 10:25:00
Archive #4 - 21/10/2020 10:20:00
Archive #5 - 21/10/2020 08:52:00
Archive #6 - 20/10/2020 13:08:00

# 3 Useful information

## 3.1 Fundamentals of Microsoft SQL Server 2019 Express

### Microsoft SQL Server 2019 Express

Microsoft SQL Server is a high-performance 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 Management Studio

The free Microsoft SQL Server Management Studio provides tools for configuring, monitoring, and managing instances or SQL servers and databases. It makes it possible to send queries and scripts to databases in the form of SQL instructions. 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 facilitates interaction with a Microsoft SQL server, including authentication and encryption of communication. After successfully logging in to the SQL server, SQL instructions can be exchanged with the server's databases using this protocol. Data are transported over TCP/IP.

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

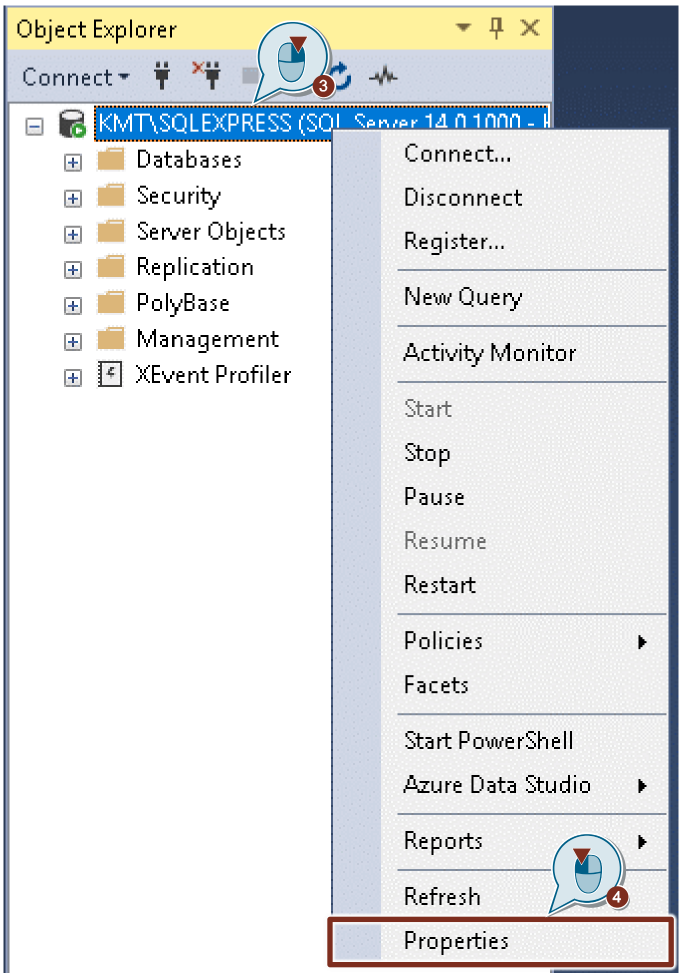
## 3.2 Settings in Microsoft SQL Server 2019 Express

### 3.2.1 SQL server login

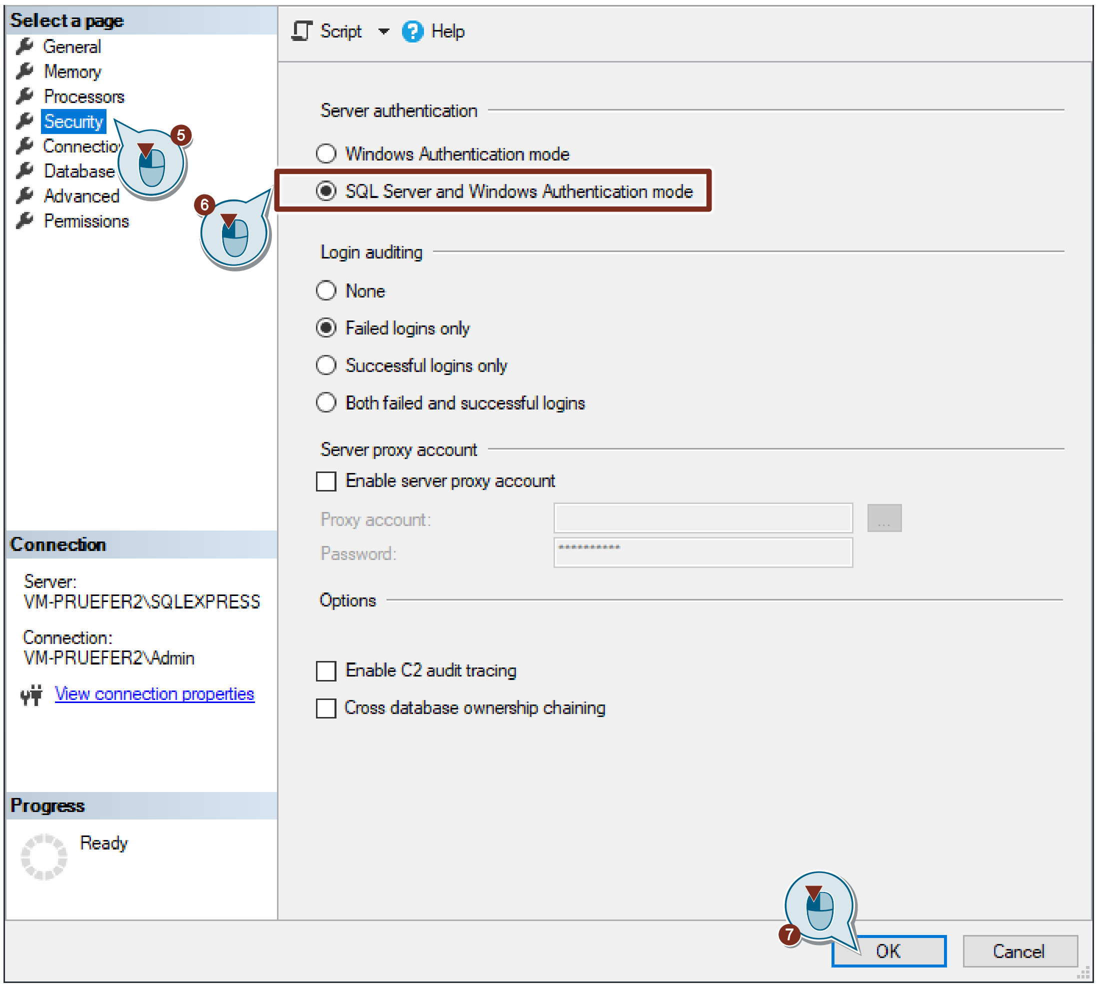
Login to a Microsoft SQL Server Express database via the SQL server authentication mode must be allowed in the database settings. This is required to log in to the database with username and password via the TDS protocol.

**Allow SQL server authentication mode**

1. Launch Microsoft SQL Server Management Studio.
2. Log on to the Microsoft SQL Server Express database using 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.



Server authentication
- [ ] Windows Authentication mode
- [✓] SQL Server and Windows Authentication mode

Login auditing
- [ ] None
- [✓] Failed logins only
- [ ] Successful logins only
- [ ] Both failed and successful logins

Server proxy account
- [ ] Enable server proxy account

Proxy account:
Password: **********

Options
- [ ] Enable C2 audit tracing
- [ ] Cross database ownership chaining

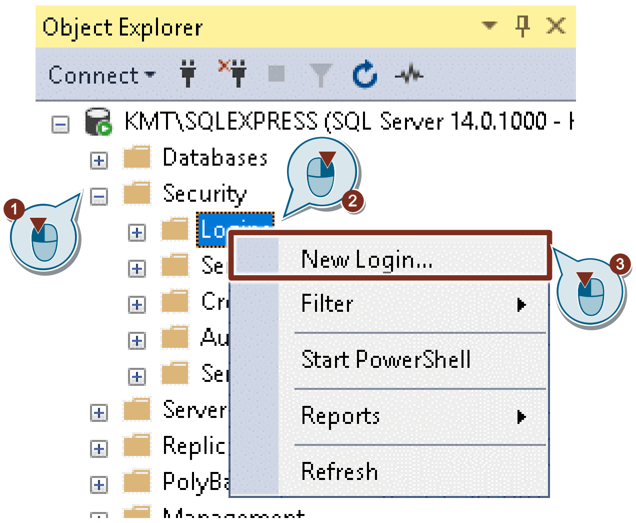
Server:
VM-PRUEFER2\SQLEXPRESS

Connection:
VM-PRUEFER2\Admin

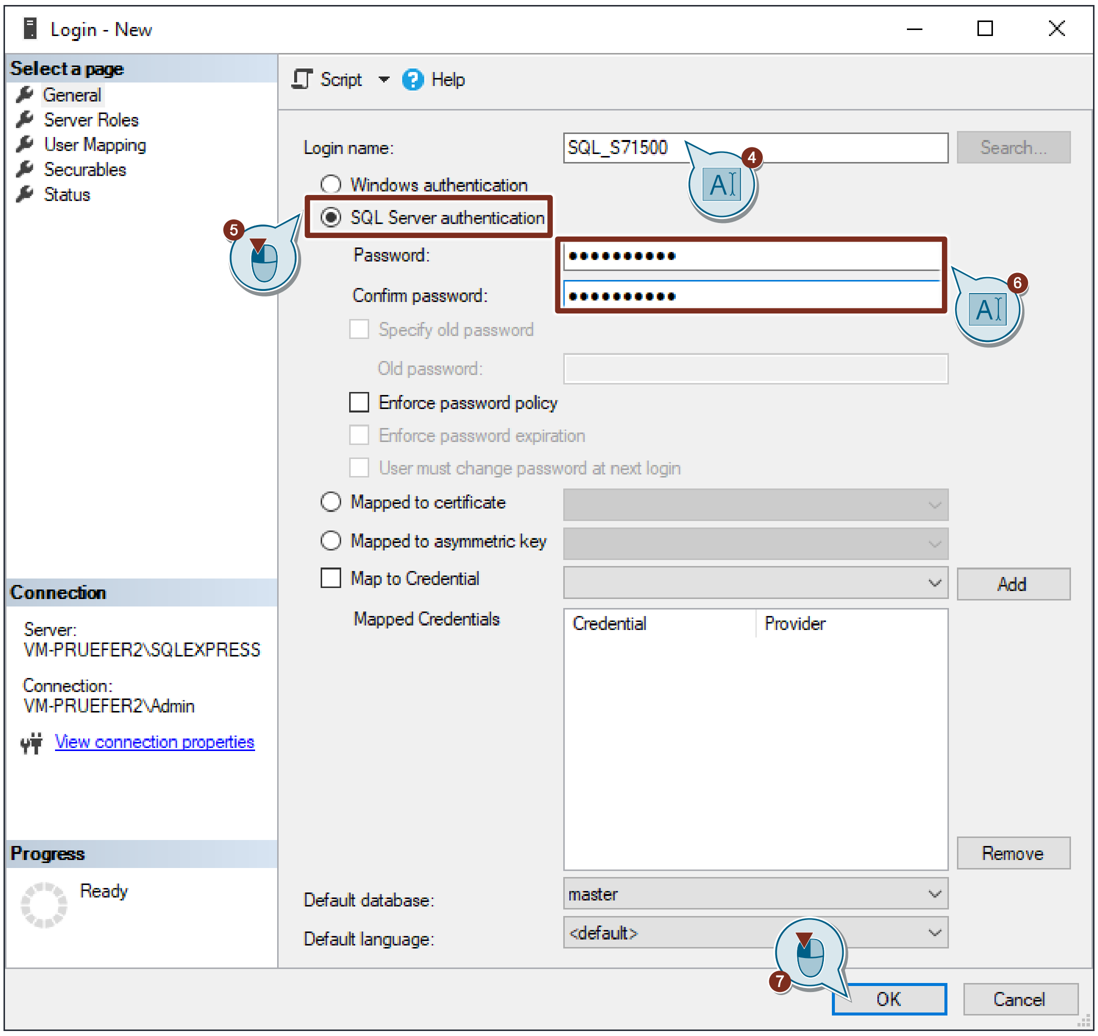
Ready

## 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. Click the "OK" button to apply the setting.


Login - New

Login name: SQL_S71500

( ) Windows authentication
(●) SQL Server authentication

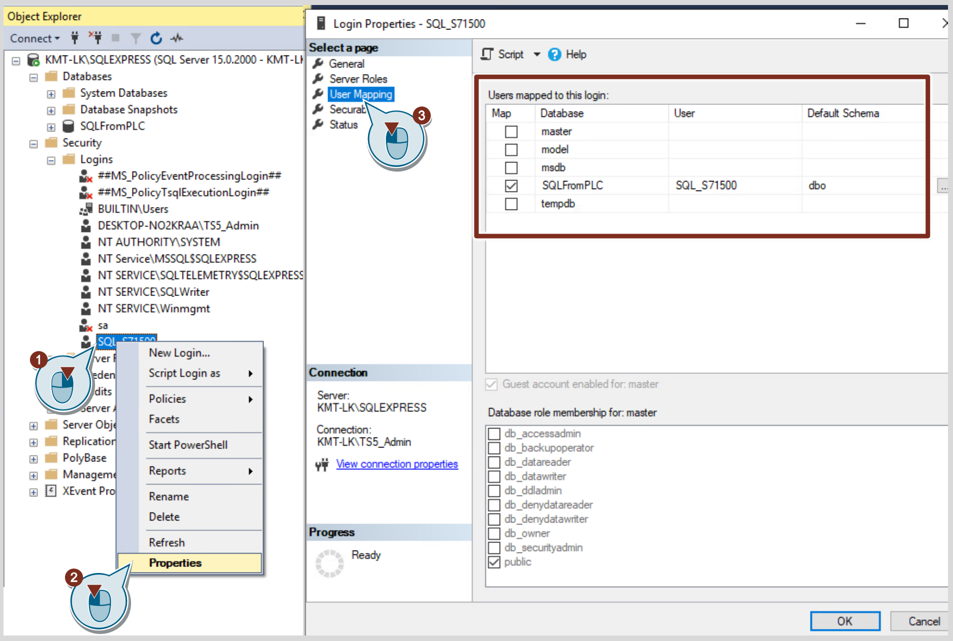
Password: ●●●●●●●●●●
Confirm password: ●●●●●●●●●●

Default database: master
Default language: <default>

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

**Note**

Makes sure that the user receives the permissions to access the database "SQLFromPLC".



KMT-LK\SQLEXPRESS (SQL Server 15.0.2000 - KMT-LK
- Databases
  - System Databases
  - Database Snapshots
  - SQLFromPLC
- Security
  - Logins
    - ##MS_PolicyEventProcessingLogin##
    - ##MS_PolicyTsqlExecutionLogin##
    - BUILTIN\Users
    - DESKTOP-NO2KRAA\TS5_Admin
    - NT AUTHORITY\SYSTEM
    - NT Service\MSSQL$SQLEXPRESS
    - NT SERVICE\SQLTELEMETRY$SQLEXPRESS
    - NT SERVICE\SQLWriter
    - NT SERVICE\Winmgmt
    - sa
    - SQL_S71500
- Server Objects
- Replication
- PolyBase
- Management
- XEvent Profiler

Server:
KMT-LK\SQLEXPRESS

Connection:
KMT-LK\TS5_Admin

Users mapped to this login:

| Map | Database | User | Default Schema |
|---|---|---|---|
| ☐ | master | | |
| ☐ | model | | |
| ☐ | msdb | | |
| ☑ | SQLFromPLC | SQL_S71500 | dbo |
| ☐ | tempdb | | |

☑ Guest account enabled for: master

Database role membership for: master

- ☐ db_accessadmin
- ☐ db_backupoperator
- ☐ db_datareader
- ☐ db_datawriter
- ☐ db_ddladmin
- ☐ db_denydatareader
- ☐ db_denydatawriter
- ☐ db_owner
- ☐ db_securityadmin
- ☑ public

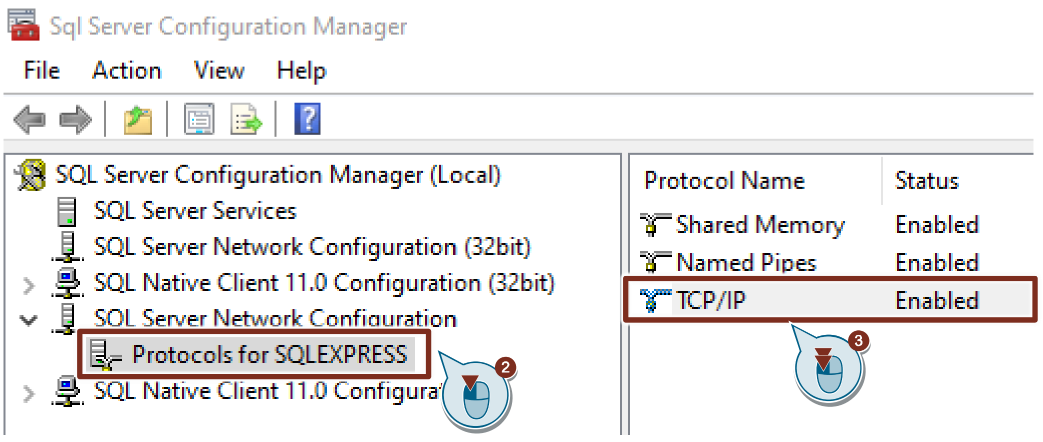
## 3.2.2 Open ports in the SQL server

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

**Note** When a firewall is active on the PC with the Microsoft SQL server database, TCP port "1433" must be allowed in the firewall for incoming connections.

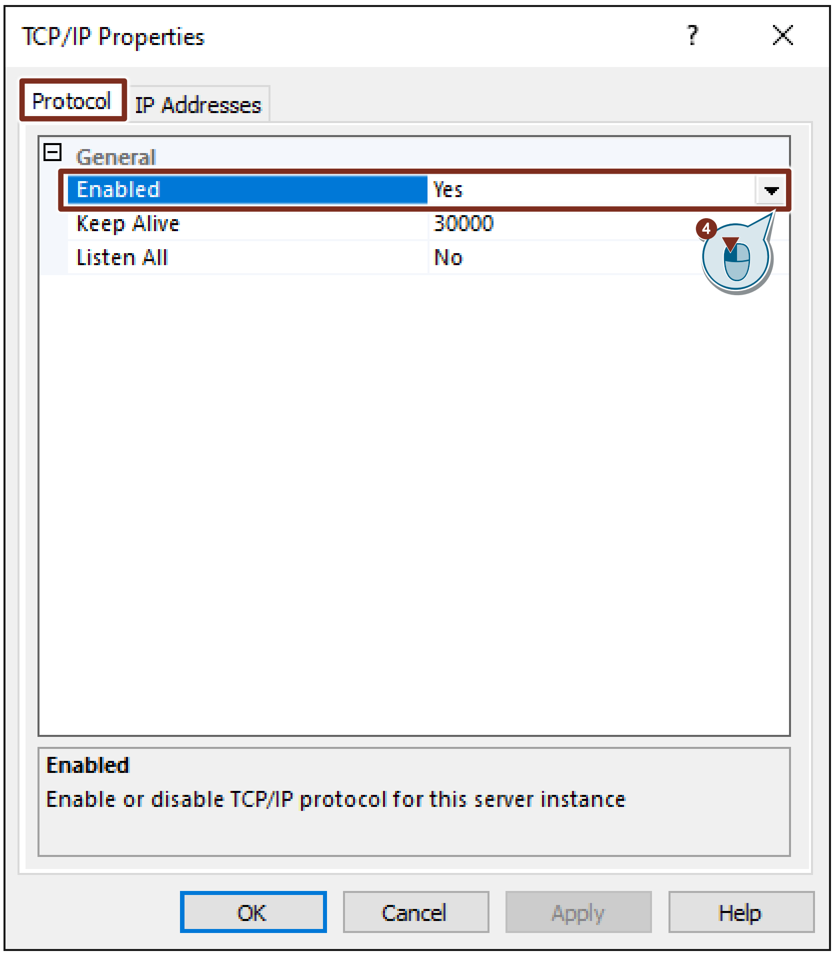
A port authorization must be set up in the SQL server so that the SQL server is reachable on the network.

1. Start "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.


Sql Server Configuration Manager
File
Action
View
Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32bit)
SQL Server Network Configuration
Protocols for SQLEXPRESS
SQL Native Client 11.0 Configura
2
3

| Protocol Name | Status |
|---|---|
| Shared Memory | Enabled |
| Named Pipes | Enabled |
| TCP/IP | Enabled |

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

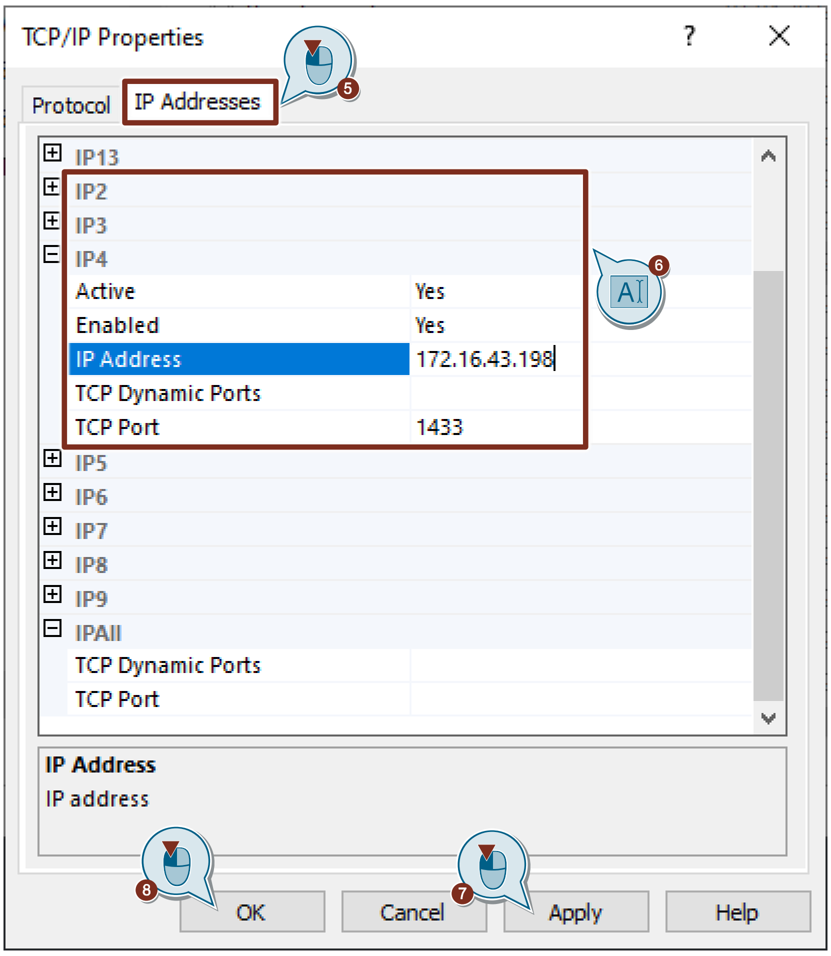


General

| | |
|---|---|
| Enabled | Yes |
| Keep Alive | 30000 |
| Listen All | No |

**Enabled**
Enable or disable TCP/IP protocol for this server instance

5. Open the "IP addresses" tab.
6. Make 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.



TCP/IP Properties

Protocol | IP Addresses

| | |
|---|---|
| IP13 | |
| IP2 | |
| IP3 | |
| IP4 | |
| Active | Yes |
| Enabled | Yes |
| IP Address | 172.16.43.198 |
| TCP Dynamic Ports | |
| TCP Port | 1433 |
| IP5 | |
| IP6 | |
| IP7 | |
| IP8 | |
| IP9 | |
| IPAll | |
| TCP Dynamic Ports | |
| TCP Port | |

**IP Address**
IP address

9. Restart the SQL server service for the changes to take effect.

## 3.2.3 Test connection to the SQL server

From a different PC, test whether Telnet establishes a connection to the IP address and port of the SQL server.

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

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

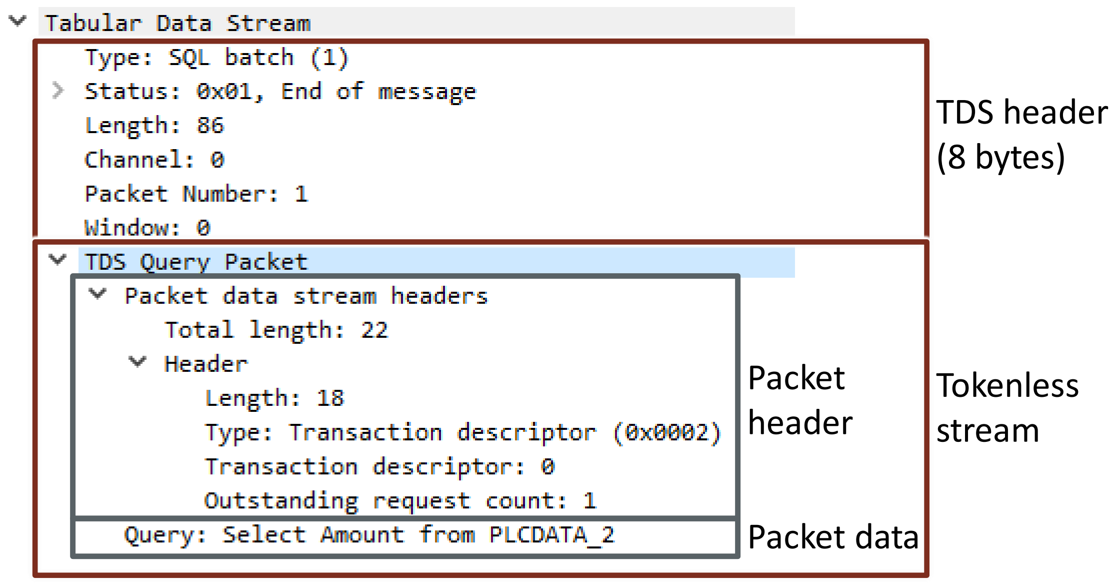
## 3.3 Tabular Data Stream (TDS) structure

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 of which is 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) in a TDS response telegram
  The DONE token marks the end of the response for each SQL instruction that is executed.
  Table 3-2 shows the structure of the DONE token.

The following Figure shows a SQL batch.

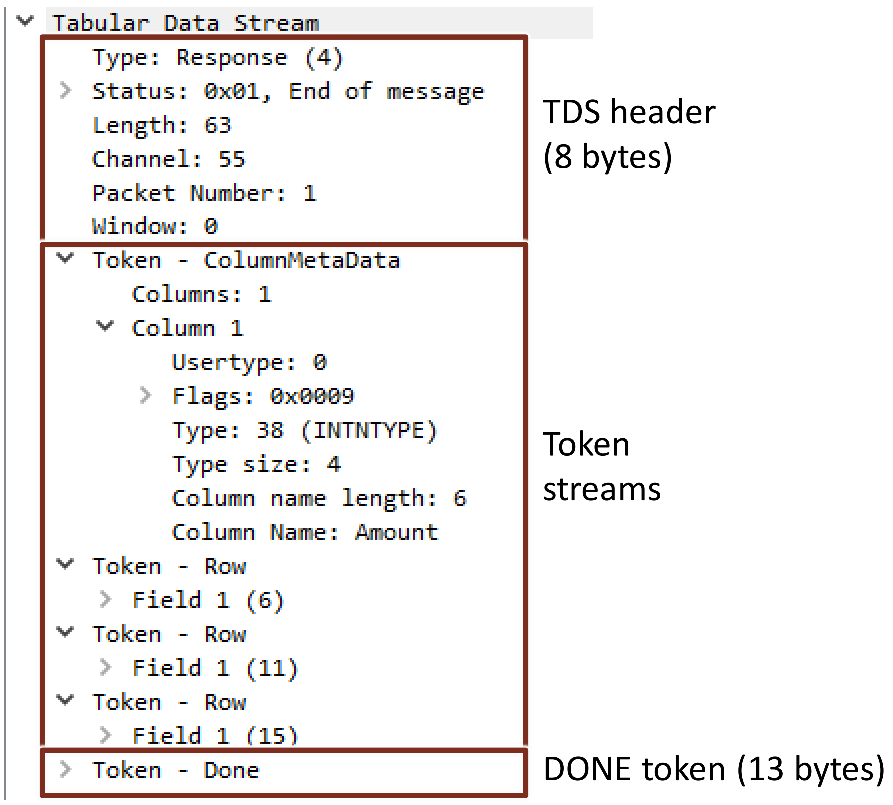
Figure 3-1



```
Tabular Data Stream
  Type: SQL batch (1)
  Status: 0x01, End of message
  Length: 86
  Channel: 0
  Packet Number: 1
  Window: 0
  TDS Query Packet
    Packet data stream headers
      Total length: 22
      Header
        Length: 18
        Type: Transaction descriptor (0x0002)
        Transaction descriptor: 0
        Outstanding request count: 1
    Query: Select Amount from PLCDATA_2
```

The following Figure shows a TDS response telegram.

Figure 3-2



```
Tabular Data Stream
    Type: Response (4)
    Status: 0x01, End of message
    Length: 63
    Channel: 55
    Packet Number: 1
    Window: 0
  Token - ColumnMetaData
      Columns: 1
      Column 1
          Usertype: 0
          Flags: 0x0009
          Type: 38 (INTNTYPE)
          Type size: 4
          Column name length: 6
          Column Name: Amount
  Token - Row
      Field 1 (6)
  Token - Row
      Field 1 (11)
  Token - Row
      Field 1 (15)
  Token - Done
```

The following Table shows the structure of the TDS header.

Table 3-1

| TDS header | Description |
|---|---|
| Type | Message type<br>1 = SQL batch<br>4 = TDS response telegram from SQL server |
| Status | Message status<br>0 = "normal" message<br>1 = End of message (EOM)<br>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 packet |
| 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<br>0x10 = DONE_COUNT, i. e. the value of "DoneRowCount" is valid. |
| CurCmd | The token of the current SQL instruction. |
| DoneRowCount | The number of rows affected by the SQL instruction. 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 instructions can easily exceed the 254-character limit. If you wish to use queries longer than 254 characters, you can 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/

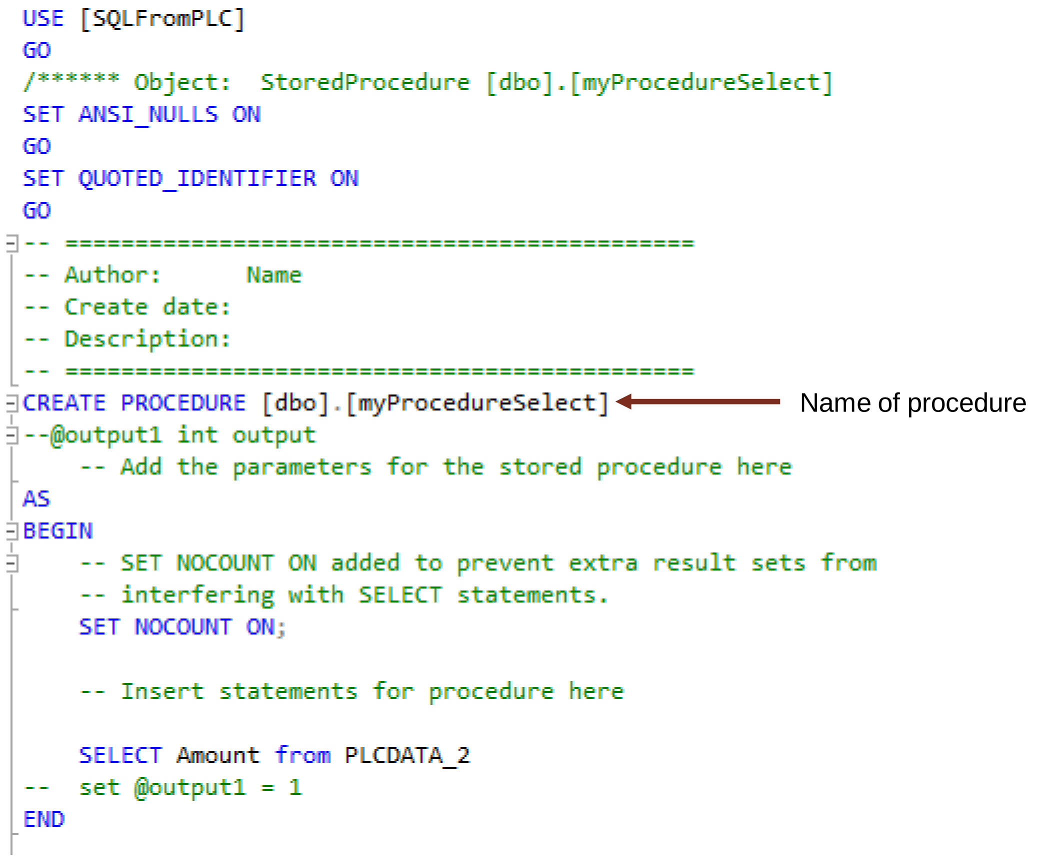
### 3.4.2 Call a stored procedure without inputs or outputs

#### 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 instruction "select" reads a row from the "Amount" column from the "PLCDATA_2" database table.

Figure 3-3



```
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
```

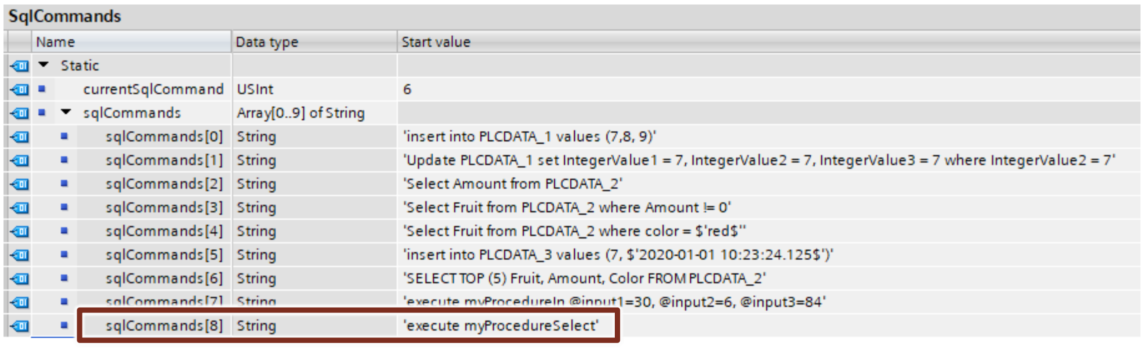
To execute the stored procedure without inputs or outputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

execute <Name of procedure>

## Read integer values

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

Figure 3-4



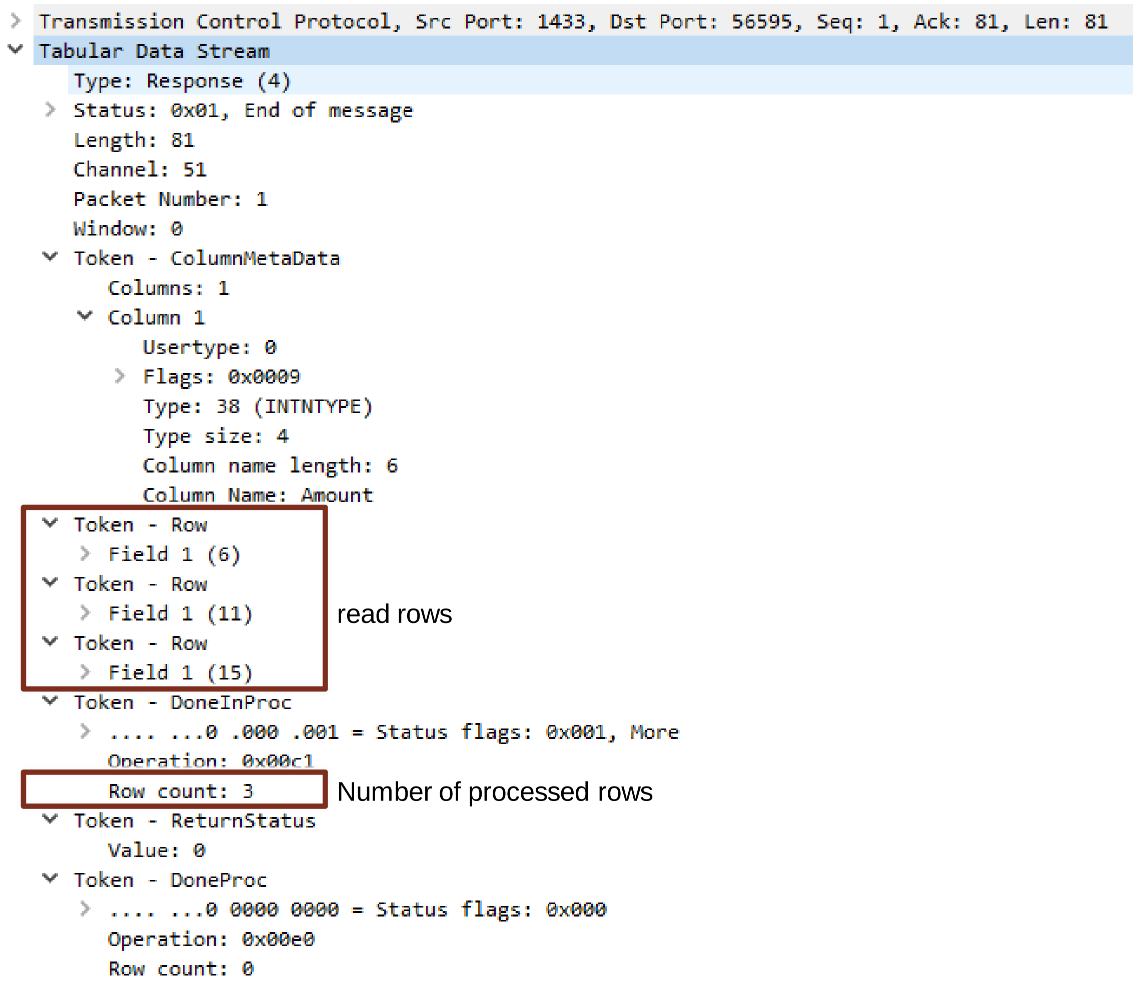
SqlCommands

| 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 3-5



```
> Transmission Control Protocol, Src Port: 1433, Dst Port: 56595, Seq: 1, Ack: 81, Len: 81
v Tabular Data Stream
     Type: Response (4)
   > Status: 0x01, End of message
     Length: 81
     Channel: 51
     Packet Number: 1
     Window: 0
   v Token - ColumnMetaData
        Columns: 1
      v Column 1
           Usertype: 0
         > Flags: 0x0009
           Type: 38 (INTNTYPE)
           Type size: 4
           Column name length: 6
           Column Name: Amount
   v Token - Row
      > Field 1 (6)
   v Token - Row
      > Field 1 (11)
   v Token - Row
      > Field 1 (15)
   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 0000 = Status flags: 0x000
        Operation: 0x00e0
        Row count: 0
```

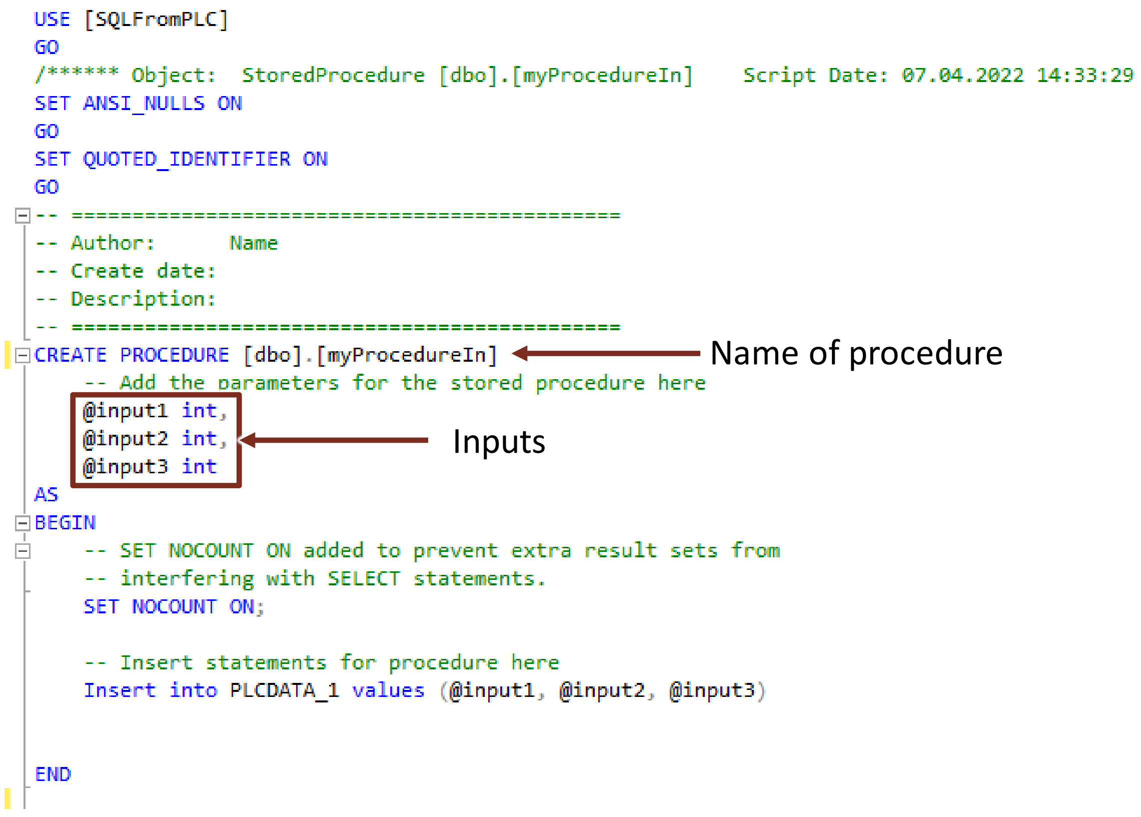
### 3.4.3 Call a stored procedure with inputs

#### Transact-SQL code of the stored procedure

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

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

Figure 3-6



```
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
```

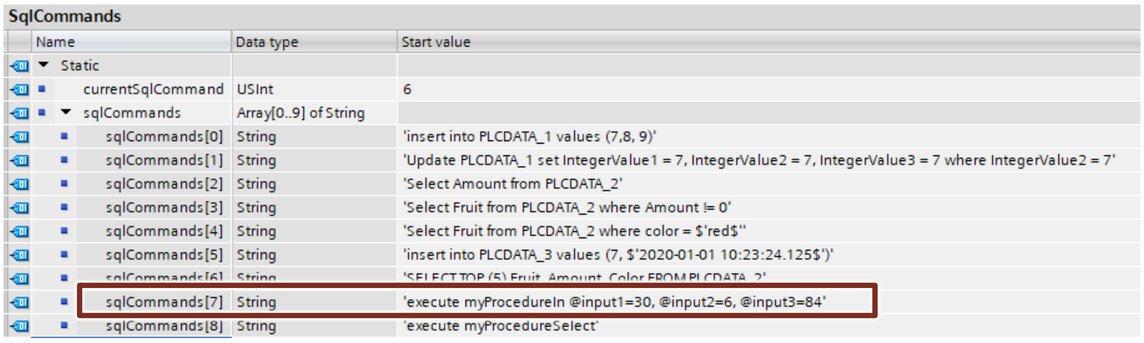
To execute the stored procedure with 3 inputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

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

## Insert integer values

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

Figure 3-7



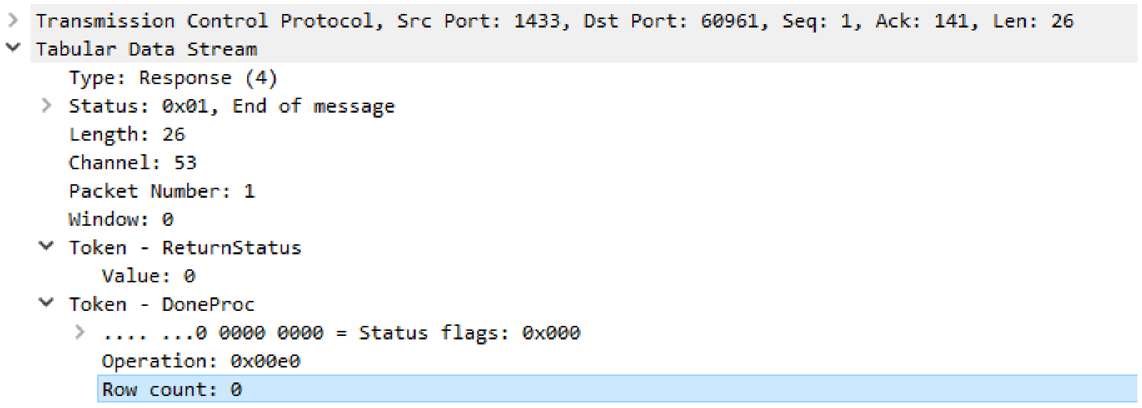
SqlCommands

| 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 with 3 inputs.

Figure 3-8



```
> 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
```

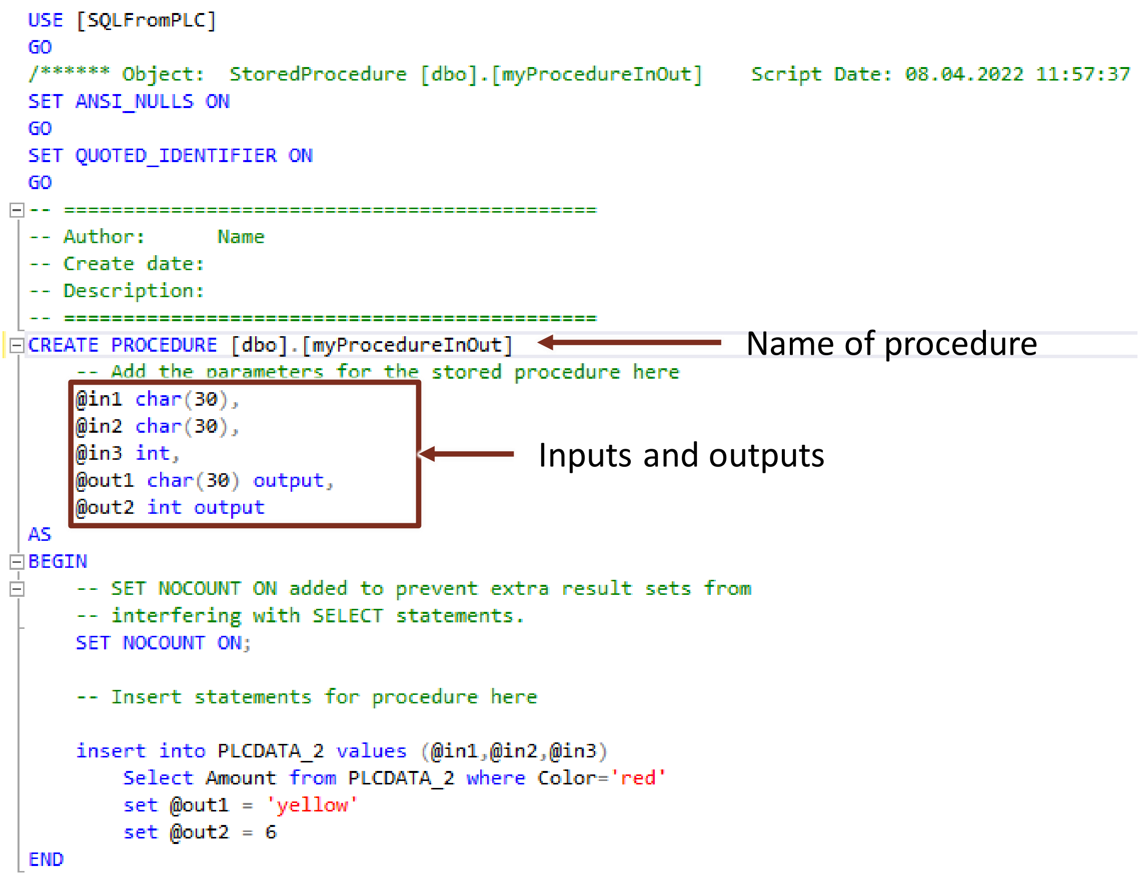
### 3.4.4 Call a stored procedure with inputs and outputs

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

Figure 3-9



```
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
```

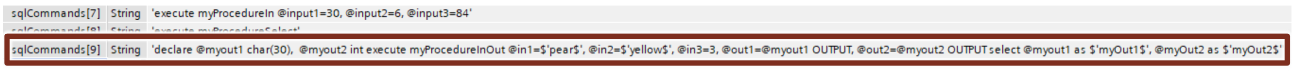
To execute the stored procedure with inputs and outputs with the FB "LSql_Microsoft", enter the following SQL command at the "command" input:

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 $'myOut2$'

## Read and insert integer values

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

Figure 3-10

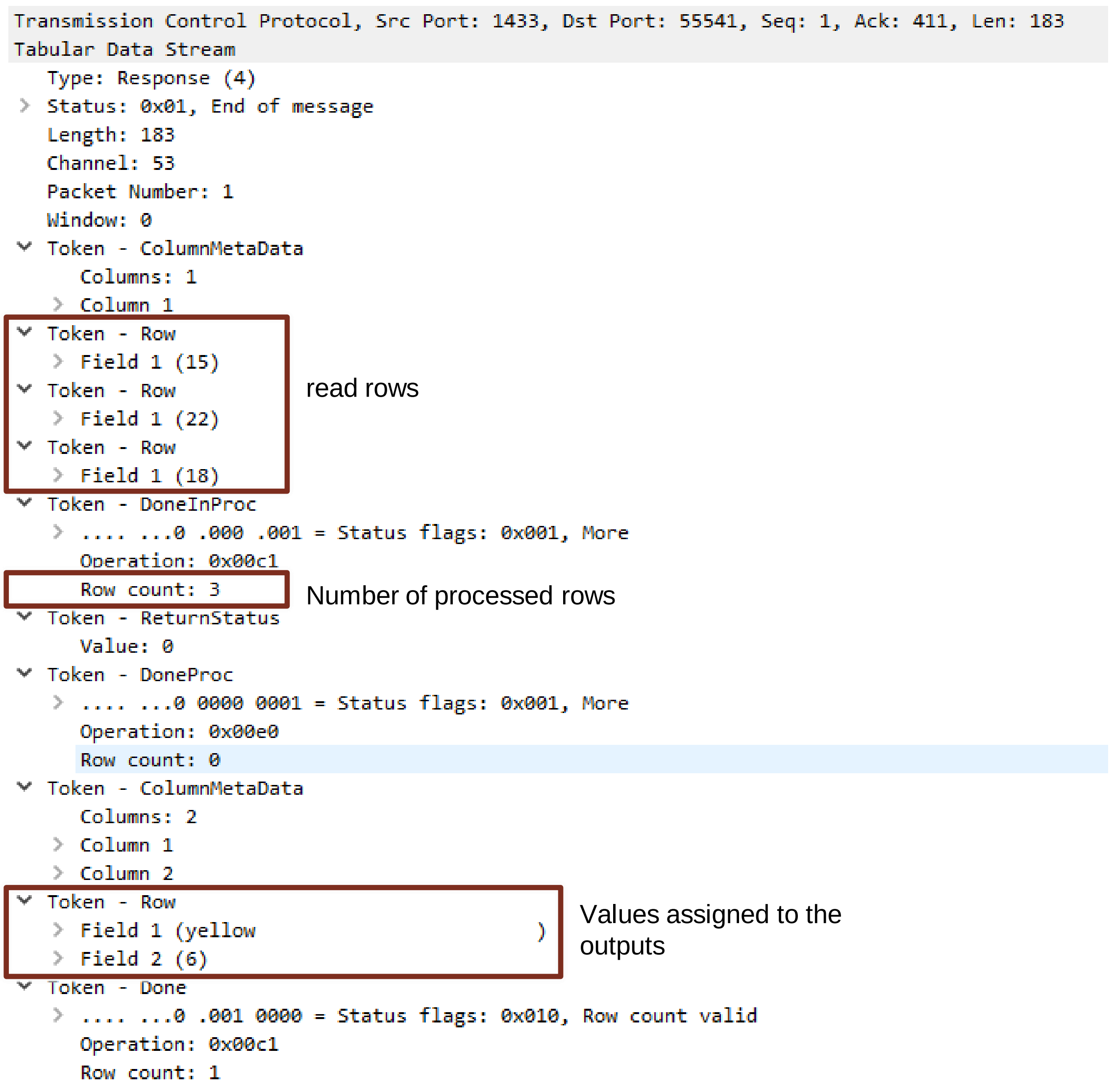


```
sqlCommands[7]  String  'execute myProcedureIn @input1=30, @input2=6, @input3=84'
sqlCommands[8]  String  'execute myProcedureSelect'
sqlCommands[9]  String  'declare @myout1 char(30),  @myout2 int execute myProcedureInOut @in1=$'pear$', @in2=$'yellow$', @in3=3, @out1=@myout1 OUTPUT, @out2=@myout2 OUTPUT select @myout1 as $'myOut1$', @myOut2 as $'myOut2$'
```

## Result

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

Figure 3-11



```
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
```

## 3.5 Modifying the FB "AnalyzeData" to analyze the TDS response telegram of a stored procedure

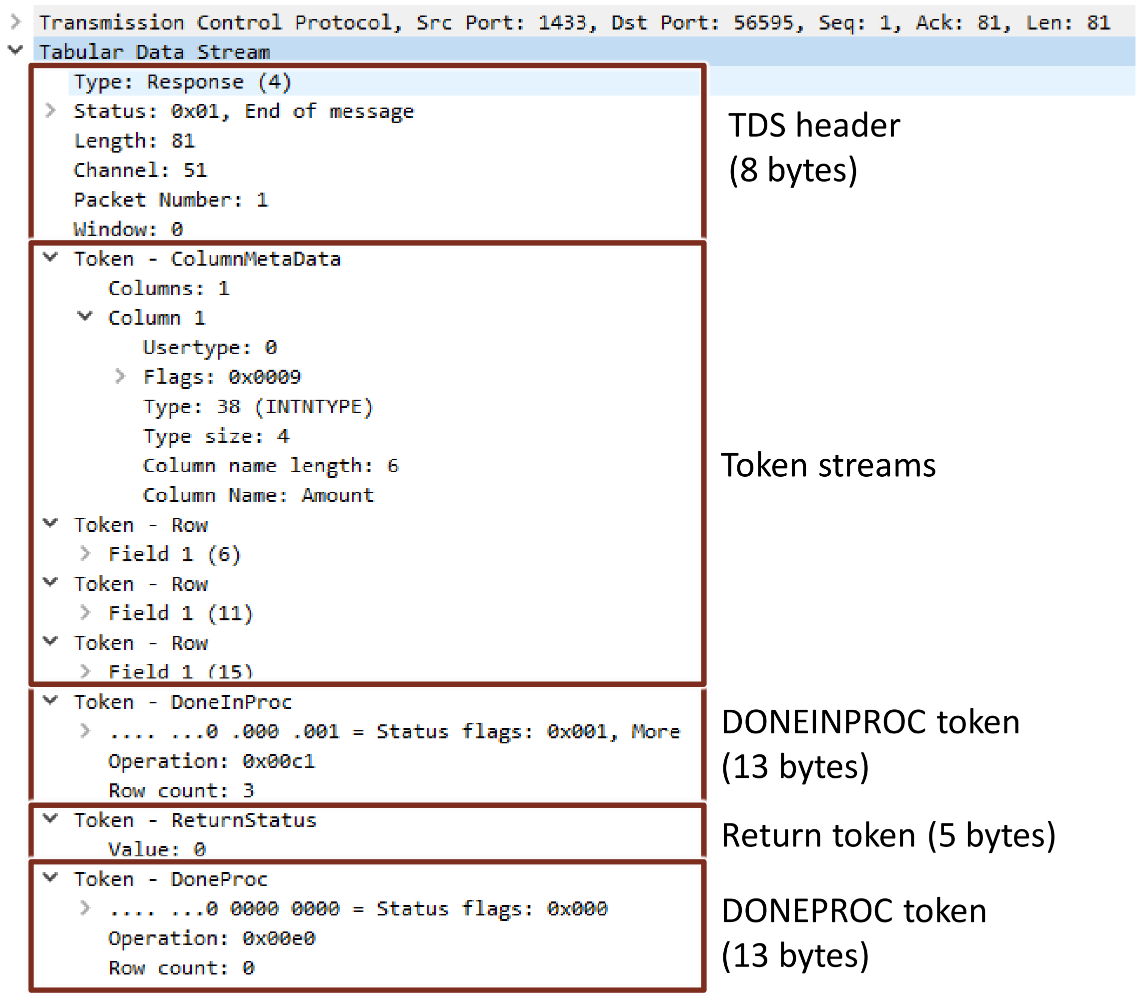
### 3.5.1 Structure of the TDS response telegram of a stored procedure

If you execute a stored procedure with a SQL instruction, the TDS response telegram is structed as follows (example):

- TDS header (8 bytes)
  Table 3-1 shows the structure of the TDS header.
- Packet data
- DONEINPROC token (13 bytes)
  A DONEINPROC token is sent for each SQL instruction that is executed within a stored procedure. The DONEINPROC token indicates that the SQL instruction is complete. The DONEINPROC token must be followed by a DONEPROC token or another DONEINPROC token.
  Table 3-3 shows the structure of the DONEINPROC token.
- ReturnStatus token (5 bytes)
  The SQL server uses this token for sending the result status value of a stored procedure that is executed via a SQL batch.
  Table 3-4 shows the structure of the ReturnStatus token.
- DONEPROC token (13 bytes)
  A DONEPROC token is sent when all SQL instructions in a stored procedure have been executed. The DONEPROC indicates the completion status of a stored procedure. A separate DONEPROC token is sent for each stored procedure that is called. The DONEPROC is also generated for stored procedures that are executed via SQL instructions. Another DONEPROC token or a DONEINPROC token can only follow a DONEPROC token if the value of the status contains the value "0x1".
  Table 3-5 shows the structure of the DONEPROC token.

The following Figure shows the TDS response telegram.

Figure 3-12

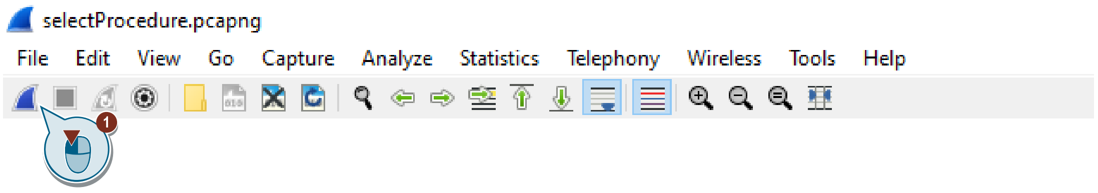


```
> Transmission Control Protocol, Src Port: 1433, Dst Port: 56595, Seq: 1, Ack: 81, Len: 81
v Tabular Data Stream
    Type: Response (4)
  > Status: 0x01, End of message
    Length: 81
    Channel: 51
    Packet Number: 1
    Window: 0
  v Token - ColumnMetaData
      Columns: 1
    v Column 1
        Usertype: 0
      > Flags: 0x0009
        Type: 38 (INTNTYPE)
        Type size: 4
        Column name length: 6
        Column Name: Amount
  v Token - Row
    > Field 1 (6)
  v Token - Row
    > Field 1 (11)
  v Token - Row
    > Field 1 (15)
  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 0000 = Status flags: 0x000
      Operation: 0x00e0
      Row count: 0
```

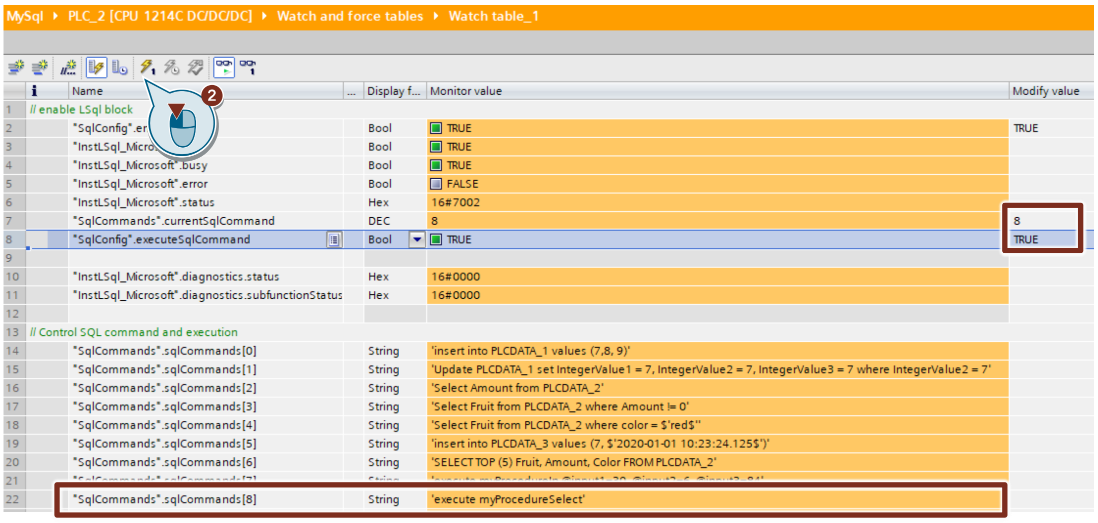
We will use Wireshark to find out how the TDS response telegram from the SQL server is structured. You can download Wireshark from the following link:

https://www.wireshark.org/download.html

1. Start a Wireshark recording.

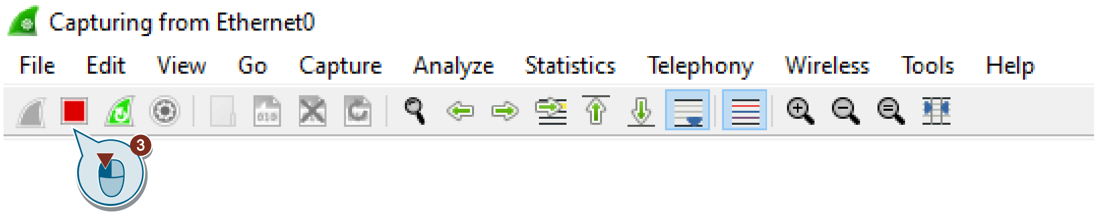

selectProcedure.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
1

2. Execute the command to run the stored procedure.

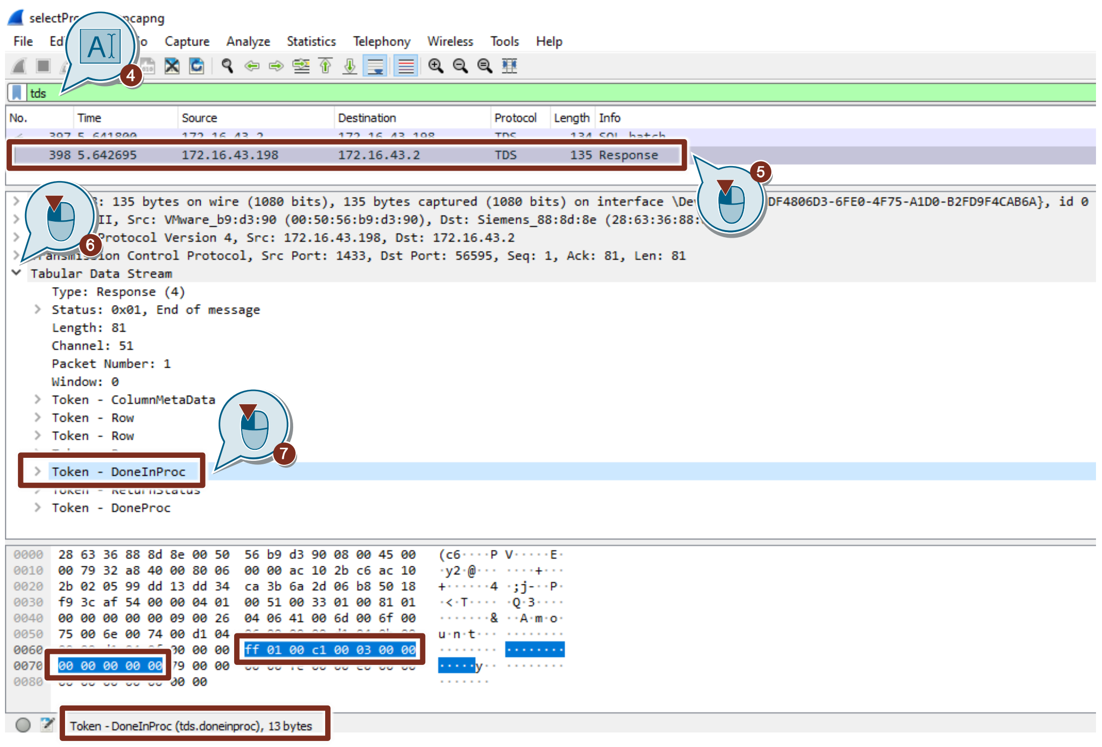

MySql ▸ PLC_2 [CPU 1214C DC/DC/DC] ▸ Watch and force tables ▸ Watch table_1
2

| | i | Name | ... | Display f... | Monitor value | Modify value |
|---|---|---|---|---|---|---|
| 1 | | // enable LSql block | | | | |
| 2 | | "SqlConfig".er | | Bool | 🟩 TRUE | TRUE |
| 3 | | "InstLSql_Micro | | Bool | 🟩 TRUE | |
| 4 | | "InstLSql_Microsoft".busy | | Bool | 🟩 TRUE | |
| 5 | | "InstLSql_Microsoft".error | | Bool | ☐ FALSE | |
| 6 | | "InstLSql_Microsoft".status | | Hex | 16#7002 | |
| 7 | | "SqlCommands".currentSqlCommand | | DEC | 8 | 8 |
| 8 | | "SqlConfig".executeSqlCommand | | Bool | 🟩 TRUE | TRUE |
| 9 | | | | | | |
| 10 | | "InstLSql_Microsoft".diagnostics.status | | Hex | 16#0000 | |
| 11 | | "InstLSql_Microsoft".diagnostics.subfunctionStatus | | Hex | 16#0000 | |
| 12 | | | | | | |
| 13 | | // Control SQL command and execution | | | | |
| 14 | | "SqlCommands".sqlCommands[0] | | String | 'insert into PLCDATA_1 values (7,8, 9)' | |
| 15 | | "SqlCommands".sqlCommands[1] | | String | 'Update PLCDATA_1 set IntegerValue1 = 7, IntegerValue2 = 7, IntegerValue3 = 7 where IntegerValue2 = 7' | |
| 16 | | "SqlCommands".sqlCommands[2] | | String | 'Select Amount from PLCDATA_2' | |
| 17 | | "SqlCommands".sqlCommands[3] | | String | 'Select Fruit from PLCDATA_2 where Amount != 0' | |
| 18 | | "SqlCommands".sqlCommands[4] | | String | 'Select Fruit from PLCDATA_2 where color = $'red$'' | |
| 19 | | "SqlCommands".sqlCommands[5] | | String | 'insert into PLCDATA_3 values (7, $'2020-01-01 10:23:24.125$')' | |
| 20 | | "SqlCommands".sqlCommands[6] | | String | 'SELECT TOP (5) Fruit, Amount, Color FROM PLCDATA_2' | |
| 21 | | "SqlCommands".sqlCommands[7] | | String | 'execute myProcedureIn @input1=30, @input2=6, @input3=84' | |
| 22 | | "SqlCommands".sqlCommands[8] | | String | 'execute myProcedureSelect' | |

3. Stop the Wireshark recording.


Capturing from Ethernet0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
3

4. Search for the response telegram from the SQL server using the filter "tds".
5. Select the TDS response telegram.
6. Open the Tabular Data Stream.
   The contents of the Tabular Data Stream appear.
7. Select the desired token.
   The length of the selected token is displayed, and the data of the selected token are highlighted.


selectPr ncapng
File Ed o Capture Analyze Statistics Telephony Wireless Tools Help
tds
4
5
6
7

```
No.   Time        Source         Destination    Protocol  Length  Info
  398 5.642695    172.16.43.198  172.16.43.2    TDS       135 Response

> : 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits) on interface \De DF4806D3-6FE0-4F75-A1D0-B2FD9F4CAB6A}, id 0
> II, Src: VMware_b9:d3:90 (00:50:56:b9:d3:90), Dst: Siemens_88:8d:8e (28:63:36:88:
> rotocol Version 4, Src: 172.16.43.198, Dst: 172.16.43.2
> ansmission Control Protocol, Src Port: 1433, Dst Port: 56595, Seq: 1, Ack: 81, Len: 81
v Tabular Data Stream
    Type: Response (4)
  > Status: 0x01, End of message
    Length: 81
    Channel: 51
    Packet Number: 1
    Window: 0
  > Token - ColumnMetaData
  > Token - Row
  > Token - Row
  > Token - DoneInProc
  > Token - DoneProc

0000  28 63 36 88 8d 8e 00 50  56 b9 d3 90 08 00 45 00   (c6···P V·····E·
0010  00 79 32 a8 40 00 80 06  00 00 ac 10 2b c6 ac 10   ·y2·@··· ····+···
0020  2b 02 05 99 dd 13 dd 34  ca 3b 6a 2d 06 b8 50 18   +······4 ·;j-··P·
0030  f9 3c af 54 00 00 04 01  00 51 00 33 01 00 81 01   ·<·T···· ·Q·3····
0040  00 00 00 00 00 09 00 26  04 06 41 00 6d 00 6f 00   ·······& ··A·m·o·
0050  75 00 6e 00 74 00 d1 04                            u·n·t··· ········
0060                 00 00 00  ff 01 00 c1 00 03 00 00   ········ ········
0070  00 00 00 00 00 79 00 00                            ·····y·· ········
0080                 00 00                               ·······

Token - DoneInProc (tds.doneinproc), 13 bytes
```

The Table below shows the structure of the DONEINPROC token.

Table 3-3

| DONEINPROC token | Description |
|---|---|
| Type | DONEINPROC token: 0xFF |
| Status | Token status:<br>• 0x1: DONE_MORE, i. e. this DONEINPROC message is not the final DONE, DONEPROC or DONEINPROC message in the TDS response telegram. More data streams will follow.<br>• 0x10 = DONE_COUNT, i. e. the value of "DoneRowCount" is valid. |
| CurCmd | The token of the current SQL instruction. |
| DoneRowCount | The number of rows affected by the SQL instruction. The value of "DoneRowCount" is only valid if the value of Status contains the value "0x10". |

The Table below shows the structure of the ReturnStatus token.

Table 3-4

| ReturnStatus token | Description |
|---|---|
| Type | ReturnStatus token: 0x79 |
| Status | Result status value of a stored procedure that is executed via a SQL batch. |

The Table below shows the structure of the DONEPROC token.

Table 3-5

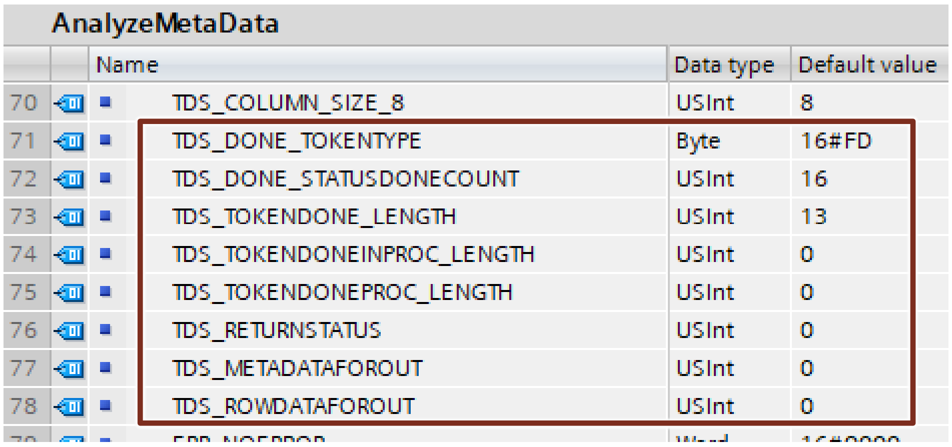
| DONEPROC token | Description |
|---|---|
| Type | DONEPROC token: 0xFE |
| Status | Token status:<br>• 0x00 = DONE_FINAL, i. e. this DONEPROC message is not the last DONEPROC message in the request.<br>• 0x1 = DONE_MORE, i. e. this DONEPROC message is not the last DONEPROC message in the TDS response telegram. More data streams will follow.<br>• 0x10 = DONE_COUNT, i. e. the value of "DoneRowCount" is valid. |
| CurCmd | The token of the current SQL instruction. |
| DoneRowCount | The number of rows affected by the SQL instruction. The value of "DoneRowCount" is only valid if the value of Status contains the value "0x10". |

## 3.5.2 Modify FB "AnalyzeMetaData"

When you execute a stored procedure with SQL instructions, it is necessary to modify the FB "AnalyzeData" to analyze the response of the SQL server to the SQL instruction you have executed.

To analyze the data from the first "Row data" token stream, modify the following constants in the FB "AnalyzeMetaData" to match the structure of the TDS response telegram.

Figure 3-13



AnalyzeMetaData

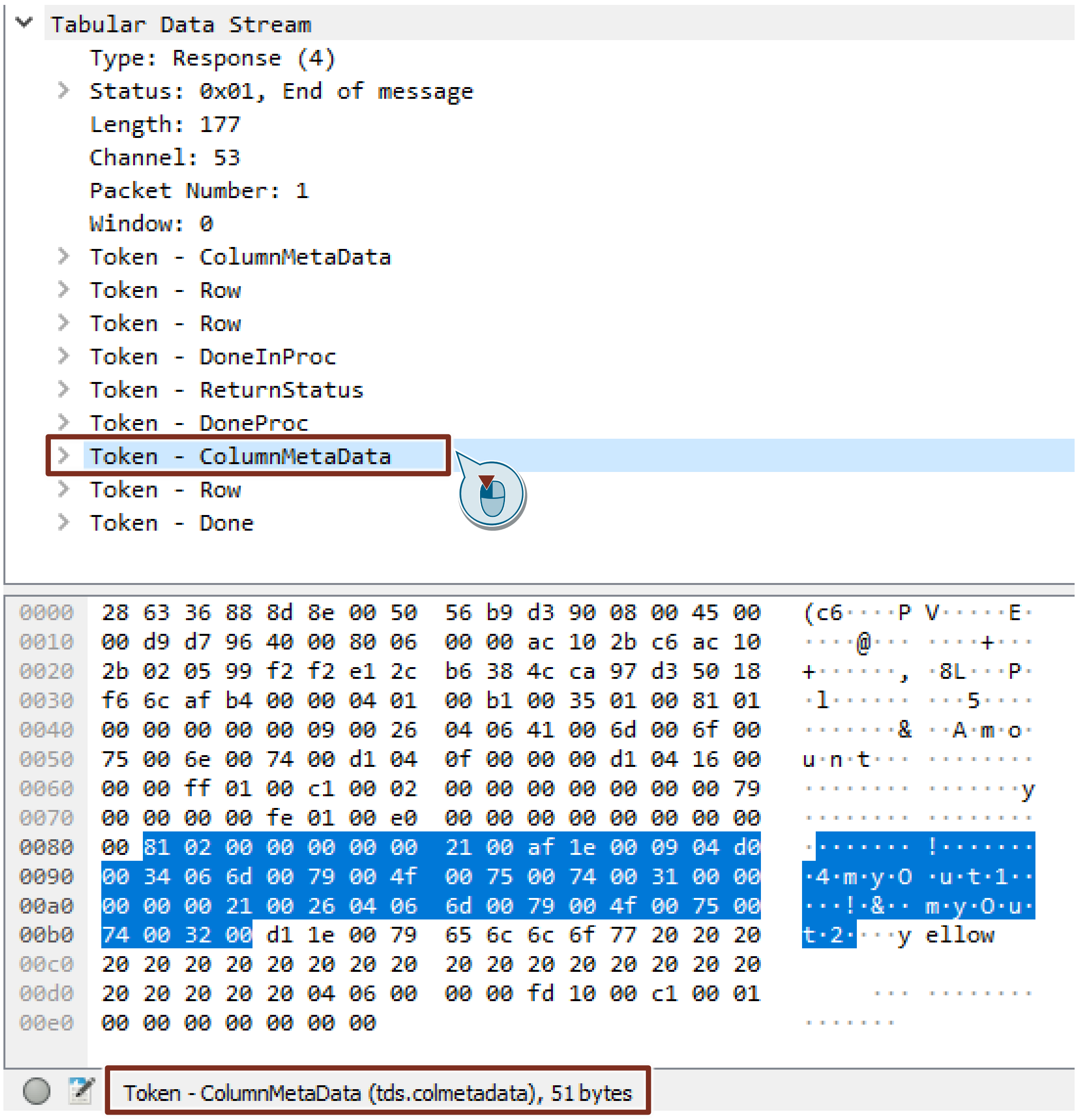
| | Name | Data type | Default value |
|---|---|---|---|
| 70 | TDS_COLUMN_SIZE_8 | USInt | 8 |
| 71 | TDS_DONE_TOKENTYPE | Byte | 16#FD |
| 72 | TDS_DONE_STATUSDONECOUNT | USInt | 16 |
| 73 | TDS_TOKENDONE_LENGTH | USInt | 13 |
| 74 | TDS_TOKENDONEINPROC_LENGTH | USInt | 0 |
| 75 | TDS_TOKENDONEPROC_LENGTH | USInt | 0 |
| 76 | TDS_RETURNSTATUS | USInt | 0 |
| 77 | TDS_METADATAFOROUT | USInt | 0 |
| 78 | TDS_ROWDATAFOROUT | USInt | 0 |

Table 3-6

| Parameters | Data type | Default value |
|---|---|---|
| TDS_DONE_TOKENTYE | Byte | Enter the type of the token that will be sent after the first "Row data" token stream when executing stored procedures, e. g. "0xFD" (DONE token) or "0xFE" (DONEINPROC token). |
| TDS_DONE_STATUSDONECOUNT | USInt | Enter the status value of the DONE token or DONEINPROC token, e. g. 16 (DONE_COUNT) if the "DoneRowCount" value is valid or 1 (DONE_MORE) if more data streams will follow. |
| TDS_TOKENDONE_LENGTH | USInt | • 13, if the DONE token is present in the TDS response telegram.<br>• 0, if the DONE token is not present in the TDS response telegram. |
| TDS_TOKEN_DONEINROC_LENGTH | USInt | • 13 * number of DONEINPROC tokens if the DONEINPROC token is present in the TDS response telegram.<br>• 0, if the DONEINPROC token is not present in the TDS response telegram. |
| TDS_TOKEN_DONEROC_LENGTH | USInt | • 13 * number of DONEPROC tokens if the DONEPROC token is present in the TDS response telegram.<br>• 0, if the DONEPROC token is not present in the TDS response telegram. |
| TDS_RETURNSTATUS | USInt | • 5 * number of ReturnStatus tokens if the ReturnStatus token is present in the TDS response telegram.<br>• 0, if the ReturnStatus token is not present in the TDS response telegram. |
| TDS_METADATAFOROUT | USInt | If multiple "ColumnMetaData" token streams are sent in the TDS response when executing a stored procedure, you have the option of entering the length (number of bytes) of the additional tokens (see Figure 3-14). |
| TDS_METAROWFOROUT | USInt | If multiple "Row data" token streams are sent in the TDS response when executing a stored procedure, you have the option of entering the length (number of bytes) of the additional tokens (see Figure 3-15). |

The following Figure shows how to find the length of the additional "ColumnMetaData" token streams.

Figure 3-14



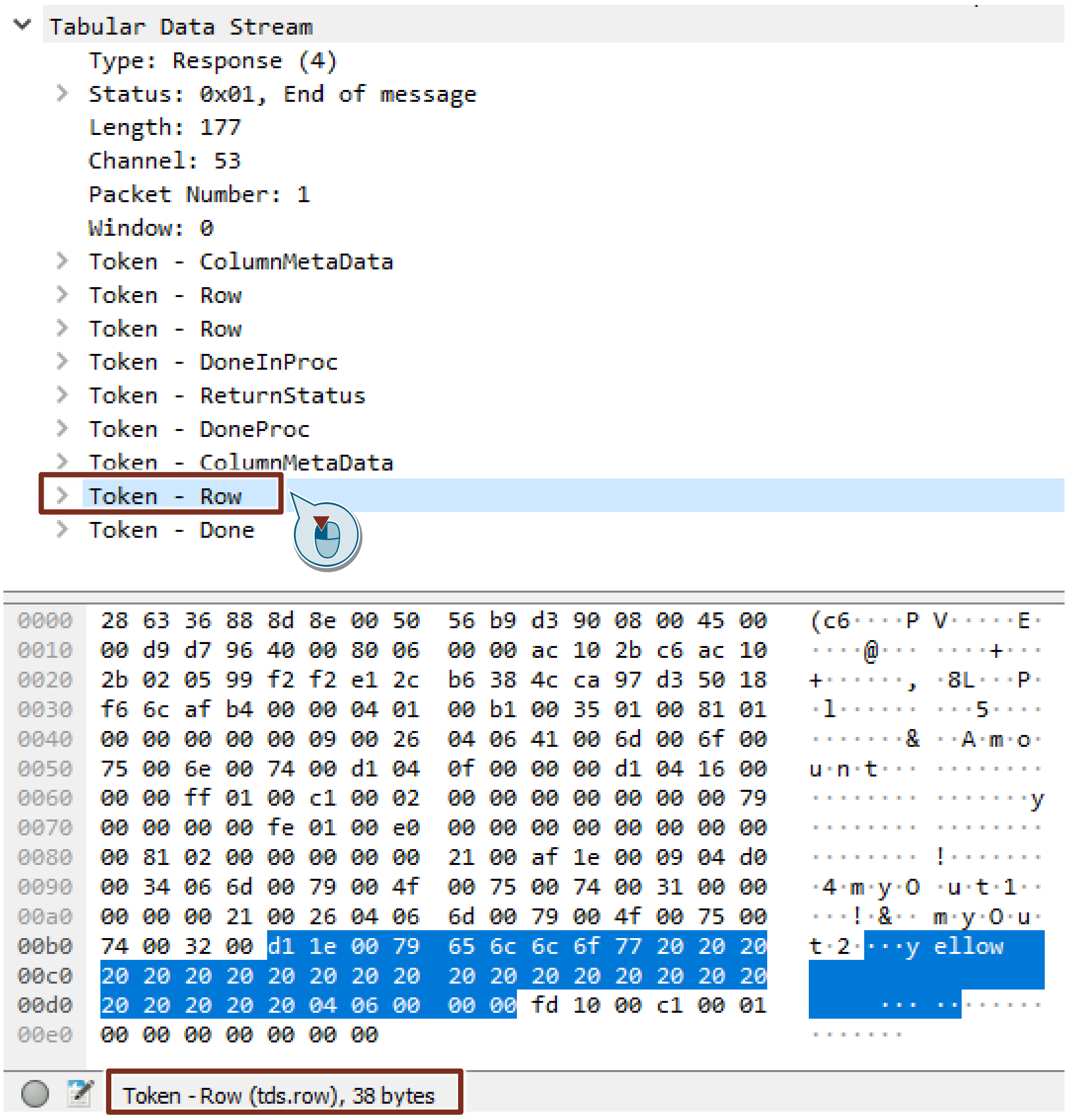
```
Tabular Data Stream
    Type: Response (4)
  > Status: 0x01, End of message
    Length: 177
    Channel: 53
    Packet Number: 1
    Window: 0
  > Token - ColumnMetaData
  > Token - Row
  > Token - Row
  > Token - DoneInProc
  > Token - ReturnStatus
  > Token - DoneProc
  > Token - ColumnMetaData
  > Token - Row
  > Token - Done
```

```
0000  28 63 36 88 8d 8e 00 50  56 b9 d3 90 08 00 45 00   (c6····P V·····E·
0010  00 d9 d7 96 40 00 80 06  00 00 ac 10 2b c6 ac 10   ····@··· ····+···
0020  2b 02 05 99 f2 f2 e1 2c  b6 38 4c ca 97 d3 50 18   +······, ·8L···P·
0030  f6 6c af b4 00 00 04 01  00 b1 00 35 01 00 81 01   ·l······ ···5····
0040  00 00 00 00 00 09 00 26  04 06 41 00 6d 00 6f 00   ·······& ··A·m·o·
0050  75 00 6e 00 74 00 d1 04  0f 00 00 00 d1 04 16 00   u·n·t··· ········
0060  00 00 ff 01 00 c1 00 02  00 00 00 00 00 00 00 79   ········ ·······y
0070  00 00 00 00 fe 01 00 e0  00 00 00 00 00 00 00 00   ········ ········
0080  00 81 02 00 00 00 00 00  21 00 af 1e 00 09 04 d0   ········ !·······
0090  00 34 06 6d 00 79 00 4f  00 75 00 74 00 31 00 00   ·4·m·y·O ·u·t·1··
00a0  00 00 00 21 00 26 04 06  6d 00 79 00 4f 00 75 00   ···!·&·· m·y·O·u·
00b0  74 00 32 00 d1 1e 00 79  65 6c 6c 6f 77 20 20 20   t·2····y ellow
00c0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
00d0  20 20 20 20 20 04 06 00  00 00 fd 10 00 c1 00 01        ··· ········
00e0  00 00 00 00 00 00 00                               ·······
```

Token - ColumnMetaData (tds.colmetadata), 51 bytes

The following Figure shows how to find the length of the additional "Row data" token streams.

Figure 3-15



```
˅ Tabular Data Stream
    Type: Response (4)
  > Status: 0x01, End of message
    Length: 177
    Channel: 53
    Packet Number: 1
    Window: 0
  > Token - ColumnMetaData
  > Token - Row
  > Token - Row
  > Token - DoneInProc
  > Token - ReturnStatus
  > Token - DoneProc
  > Token - ColumnMetaData
  > Token - Row
  > Token - Done

0000  28 63 36 88 8d 8e 00 50  56 b9 d3 90 08 00 45 00   (c6···P V·····E·
0010  00 d9 d7 96 40 00 80 06  00 00 ac 10 2b c6 ac 10   ····@··· ····+···
0020  2b 02 05 99 f2 f2 e1 2c  b6 38 4c ca 97 d3 50 18   +······, ·8L···P·
0030  f6 6c af b4 00 00 04 01  00 b1 00 35 01 00 81 01   ·l······ ···5····
0040  00 00 00 00 00 09 00 26  04 06 41 00 6d 00 6f 00   ·······& ··A·m·o·
0050  75 00 6e 00 74 00 d1 04  0f 00 00 00 d1 04 16 00   u·n·t··· ········
0060  00 00 ff 01 00 c1 00 02  00 00 00 00 00 00 00 79   ········ ·······y
0070  00 00 00 00 fe 01 00 e0  00 00 00 00 00 00 00 00   ········ ········
0080  00 81 02 00 00 00 00 00  21 00 af 1e 00 09 04 d0   ········ !·······
0090  00 34 06 6d 00 79 00 4f  00 75 00 74 00 31 00 00   ·4·m·y·O ·u·t·1··
00a0  00 00 00 21 00 26 04 06  6d 00 79 00 4f 00 75 00   ···!·&·· m·y·O·u·
00b0  74 00 32 00 d1 1e 00 79  65 6c 6c 6f 77 20 20 20   t·2····y ellow   
00c0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20                    
00d0  20 20 20 20 20 04 06 00  00 00 fd 10 00 c1 00 01        ··· ········
00e0  00 00 00 00 00 00 00                               ·······

Token - Row (tds.row), 38 bytes
```

# 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:

support.industry.siemens.com/cs/my/src

### 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 Links and literature

Table 4-1

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

## 4.3 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 |