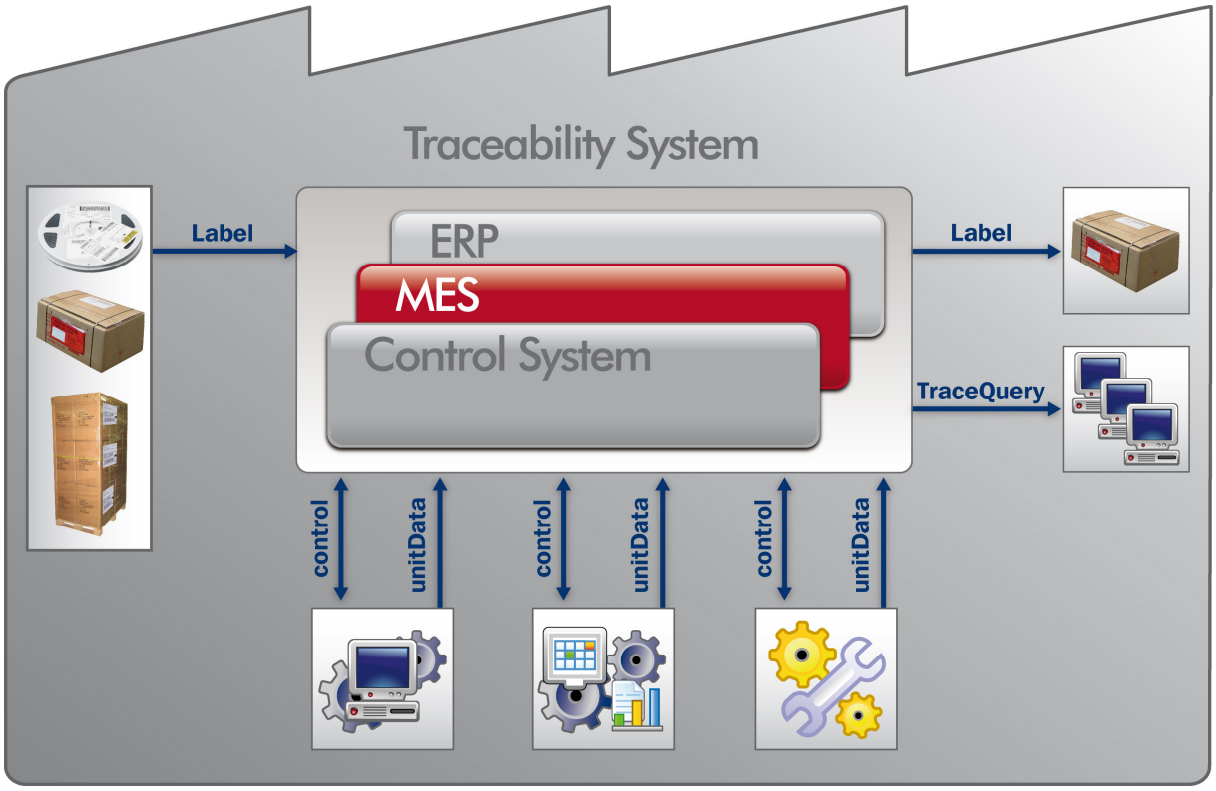
Electronic Components and Systems

# Identification and Traceability in the Electrical and Electronics Industry




## ZVEI Interfaces to Shopfloor

# Transfer Protocols

Version 1.1.0

# Foreword

Work on the ZVEI manual for the entire supply and value-added chain (see MIT 1 "Guideline for Identification and Traceability) has also served as a draft for an interface to Shopfloor for connecting machines, devices and workstations.

One goal of this undertaking is to standardise the interface for processes in general.

The result was the creation of two XML-based interfaces which are freely available and recommended by ZVEI for connecting to Shopfloor:

- ***control*** for transferring data (requests and return messages) for advanced process control while a product is being processed
- ***unitData*** for transferring processing data of a product

# History / changes

In MIT-2 "ZVEI-Interfaces-ChangeHistory" the history of changes of the interfaces ***control*** and ***unitData*** is described.

ZVEI – German Electrical and Electronic Manufacturers' Association e.V.

Electronic Components and Systems

Lyoner Straße 9

60528 Frankfurt am Main

Phone: 069 6302 – 276

Fax: 069 6302 – 407

E-mail: zvei-be@zvei.org

www.zvei-traceability.de

# Table of contents

# 1 Introduction

This document contains descriptions of the transfer protocols used for transferring xml structures of the ZVEI standard interfaces..

## 1.1 Symbols used

Three different symbols are used in this documentation to emphasise important content items.

**Attention!**

This symbol refers to important information for which compliance is absolutely mandatory.

**Explanation!**

This symbol refers to explanatory information.

**Tip!**

This symbol identifies tips which provide faster or more efficient solutions.

## 1.2 Legend

**[** Node/attribute **]** Square brackets: ➔ optional node/attribute

**<** Node/attribute **>** Pointed brackets: ➔ alternative node/attribute

**Explanation!**

If an attribute is required (not optional), the value must be assigned (no empty string).

If an attribute is not required (optional) but is present with the value = "" (empty string), the attribute will be ignored. The attribute will then be handled as if it were not even present.

# 2 Transfer protocols

This chapter describes the transfer protocols for transferring XML structures. There are several transfer protocols available for transferring XML structures.

## 2.1 File

This section describes the requirements and limitations of using a file interface for transferring XML structures.

### 2.1.1 File naming convention

The file name of each XML file must be unique. In order to minimise access to the hard drive, the user must be able to obtain important selective information simply from the file name without needing to open the file.

For this reason, the file name comprises the following fields:

**[Type_]Reference_Timestamp[_Counter].xml**

#### 2.1.1.1 [Type_]

The optional type field in the file name may be used to identify the nature of the transferred data.

Examples:

'R_' for request with a bidirectional connection
'A_' for answer with a bidirectional connection
'PD_' for process data

**Attention!**

If the different types of XML structures (e.g. various ZVEI standard interfaces or bidirectional connections with request and reply) use the same communication directory for exchanging information, then an indicator should be present in this particular part of the file name.

#### 2.1.1.2 Reference

The file name should refer to the contents of the file. This could, for example, be a serial number or a machine, order or sensor number.

### 2.1.1.3 Timestamp

The uniqueness of a file name is initially guaranteed through the use of a timestamp and optionally through a counter. The format of the timestamp must be considered. Multiple files must not be written within the smallest unit of time of the chosen format.

**Timestamp examples:**

- YYYYMMDD_hhmmss (YearMonthDay_HourMinuteSecond
- YYMMDD_hhmmssSSS (YearMonthDay_HourMinuteSecondMilliseconds)

**Attention!**

The annual daylight saving time change from summer to winter must be taken into account. This can be done by including a counter *n* in the file name or by checking the existing file names.

### 2.1.1.4 [_Counter]

The file name may optionally include a counter.

**Attention!**

When multiple files are written within the smallest unit of time of the timestamp, then the uniqueness of the files can no longer be guaranteed. In this case, the uniqueness can be ensured through the use of a counter.

### 2.1.1.5 Sample file names

A few examples include:

- Reference_timestamp.xml
  Serialnumber_YYYYMMDD_hhmmss.xml
  Example: 4711001_20090502_080000.xml
- Type_Reference_timestamp.xml
  Request_Ordernumber_YYYYMMDD_hhmmss.xml
  Example: R_4712001_20090502_080000.xml
- Type_Reference_timestamp_counterxml
  Processdata_Sensornumber_YYYYMMDD_hhmmss_counter.xml
  Example: PD_4713001_20090502_080000_12.xml

**Explanation!**

The naming convention was not used for sample files because it should be possible to recognise the relevant interfaces and their version based on the file names.

### 2.1.2 File handling

As with all ZVEI standard file interfaces, the transfer of the file also results in a change of responsibilities:

- The sender of the file must write the file prior to the transfer, close all file channels and release any file locks
- The sender may not open the file or lock it once the transfer completed
- The receiver may not open or lock the file prior to completion of the transfer
- The receiver is responsible for archiving and deleting the file

The file is transferred by moving it to a transfer directory. The following scenarios are supported:

1) Sender and receiver are located on the same computer and use a local file system for the transfer
2) Sender and receiver are located on different computers and use a file system local to the sender for the transfer
3) Sender and receiver are located on different computers and use a file system local to the receiver for the transfer

When sender and receiver are located on different computers, then option 2 above is the preferred choice. This is due to the fact that in the case of a network failure, the sender would otherwise have to provide a buffering concept for the data that cannot be written over the network.

Adherence to the following procedure is required in order to guarantee a smooth transfer:

a) The sender first writes a file to a working directory. This file is then moved to a transfer directory **in the same file system!** This is the only way to ensure that the move operation is **atomic,** thus preventing mutual blocking.

b) Once the receiver recognises a file in the transfer directory, processing of the file begins. The original file is archived or deleted depending on the requirements. The deletion of the original file from the transfer directory can occur at different times, depending on requirements:

   - If the process is asynchronous (no confirmation regarding the successful processing is necessary), then the original file can be deleted at the start of processing. This is a good solution when the transfer directory is not local to the receiver. (Otherwise after a network failure a check must determine whether the file has already been processed.)
   - If the process is synchronous (confirmation regarding the successful processing is necessary), the original file may only be deleted after processing has been successfully completed. In this case the deletion signifies the conclusion of the processing.

Comment re. b):

When the transfer directory is local to the sender (scenario 2 above), the following courses of action are to be followed:

   - Using asynchronous processing, the original file is moved to a local working directory at the beginning of the processing. From here it is processed and on successful conclusion either archived or deleted.
   - Using synchronous processing, the original file is copied to a working directory local to the receiver and then processed from this location. Once the processing is successfully completed, the original file will be deleted and the local copy then archived or deleted.

**Attention!**

Tip regarding validity of data:

After a network failure or receiver downtime, data may be out of date. The conditions regarding the expiry of data and the necessary handling must be individually specified. The same applies to the expiry of timeouts during the handshake between the communication partners.

**Attention!**

Error handling**:**

The sender is exclusively responsible for error handling until the interface files are located in the transfer directory.

The responsibility for error handling from the moment the file is located in the transfer directory lies with the receiver.

Error handling should include a logging and reporting mechanism for errors.

**Attention!**

Configuration**:**

All communication and archive directories of the communication partners must be configurable.

## 2.2 TCP/IP

This section describes the requirements and limitations of using a TCP/IP interface for transferring XML structures.

### 2.2.1 Workflow

The IP address of the server and the port number for the connection must be configurable on the client side for the transfer of XML structures using a TCP/IP interface.

Communication data is transferred in a pure byte stream.

The following sequence diagram illustrates a simple communication process between a client and a host. The client tries to connect to a host over a known port. If the connect is successful, the client sends the data as a byte stream to the host.

The host reads the data from the byte stream and transfers it to an XML parser. The parser validates the data using the schema file that is referenced in the byte stream and subsequently processes the data.

**Explanation!**

In order to validate the data and to identify the sent data packet type, the XML stream must include a valid reference in the form of a URL in the defined schema file. See the specification of the relevant XML format.

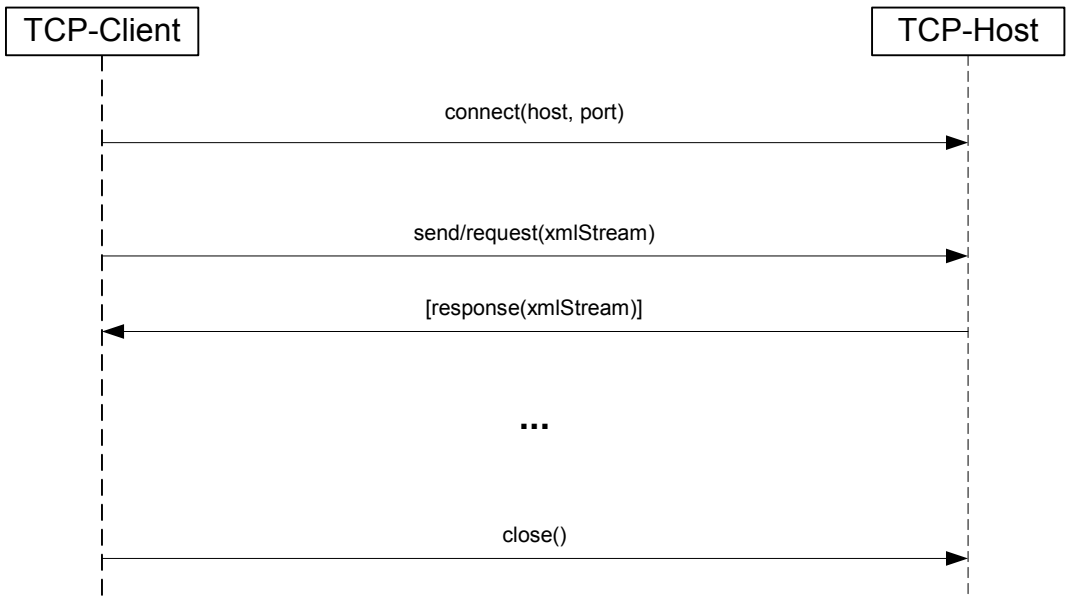



Illustration 1: Typical sequence for the transfer of an XML structure

### 2.2.2 Monitoring of the TCP connection

When an existing TCP connection breaks down, the operating system does not always reliably report that this has happened. In such situations the communication partners may be unaware that no more data is being sent over a connection.

Several methods for the detection of inactive connections may be used to prevent broken connections from remaining open for an extended time.

#### 2.2.2.1 Timeout

A maximum amount of time can be configured for which a connection may remains open without receiving data. If the time since the last request exceeds this amount, the connection will be closed.

#### 2.2.2.2 Keepalive

A keepalive message can be sent via the TCP connection at regular intervals to test whether it is still open. If an error occurs while the keepalive message is being sent, the connection is closed.

The content and frequency of the keepalive message can be configured. The communication partner must ignore this test message.

#### 2.2.2.3 Number of connections per IP address

In most applications there is only one active TCP connection to a TCP host per workstation.

When opening a new connection the existing connections can be checked to detect stale connections. If there is an existing connection to the same workstation, it will be terminated before the new connection will be opened.

# 3 Appendixes

## 3.1 Index of documents

| | |
|---|---|
| MIT 1 "Guideline for Identification and Traceability" | ZVEI Guideline for the entire supply and value chain |
| MIT-2 "ZVEI-Interfaces-ChangeHistory" | This document describes the history of changes to the interfaces control and unitData. |

## 3.2 Index of illustrations



## 3.3 Index of relevant terminology and abbreviations

| Term | Description |
|---|---|
| Sender | Communication partner who provides files |
| Receiver | Communication partner who receives the files and processes them further |
| Communication directory | File directory through which the communication partners exchange files containing data |
| Node | Node in the xml tree structure |
| Subnode | Node in the xml tree structure below a node |