VDMA Documents
Food Processing Machinery and Packaging Machinery

Guideline:

Integration of intelligent components in packaging and processing machinery in a GxP environment

- Requirements arising from 21 CFR 11
- Structuring of data interchange to take account of the requirements of 21 CFR 11
- Standardized communication (VDMAXML_P Version 2.0)
- Examples of the implementation of GxP-specific services

No. 9/2004, revised edition 2012
November 2012

Verband Deutscher Maschinen- und Anlagenbau e.V.
(German Engineering Federation)

Fachverband Nahrungsmittelmaschinen und Verpackungsmaschinen
(Food Processing and Packaging Machinery Association)
Chairman: Volker Kronseder
Managing Director: Richard Clemens

Lyoner Straße 18
D-60528 Frankfurt am Main
Tel.: +49 69 66 03-14 31
Fax +49 69 66 03-12 11
e-mail nuv@vdma.org
Internet www.vdma.org/packtech

VDMA
Technology serving people

Contents

This publication has been drawn up by Working Party "21 CFR Part 11 - Interfaces" in the VDMA's "Pharmaceutical and Cosmetic Machinery" Sector. It is available for downloading from WWW.VDMA.ORG/PACKTECH. Suggestions for improvements and additions may be sent to the following address: Fachverband Nahrungsmittelmaschinen und Verpackungsmaschinen im VDMA, Lyoner Straße 18, 60528 Frankfurt/M., Fax: +49 69/6603-1211.

Our special thanks go to Mr. Franz Vater whose preliminary work provides the basis for the material presented in Section 6.

Preliminary remark on the 2012 revision

This VDMA Publication was revised in 2012. As compared with the first edition of 2004, the following modifications have been made:

- Editorial revision of the Introduction, aiming at attaching a higher priority to the core topic of the publication — namely integration of components in machinery.
- Shifting of the specified services 'Login/Logout'; 'Change of password', 'Format', 'AuditTrail', 'EventLog' and 'BatchControl' to the Appendix of the publication. This is to illustrate that these specified services do not belong to the mandatory part of the specification of the communication protocol VDMAXML_P, resulting in the fact that, in the case of components supporting VDMAXML_P, no support for these services can be be presupposed, but that their implementation may be subject to special agreement.
- The status of attribute 's' – memory category – is changed from 'mandatory' to 'optional'. This is to account for the practice of implementation.
- The version number of VDMAXML_P is set to 2.0. This version is compatible with version 1.0.

1. Introduction

In 21 CFR, Part 11, the American regulatory agency, the FDA, has set out its requirements for the use of electronic documents and electronic signatures in the pharmaceutical industry. This also has consequences for the supplying industries. Thus, it is expected of the manufacturers of packaging and processing machines for the pharmaceutical industry that the machines meet the requirements of 21 CFR, Part 11. In implementing these requirements, the integration of intelligent components assumes a position of key significance. This being so there are fundamentally two types of integration to be distinguished: integration in a machine and integration in a production or packaging line. These types are illustrated schematically in Figures 1 and 2.

Fig. 1: Scheme for integrating intelligent components in machines

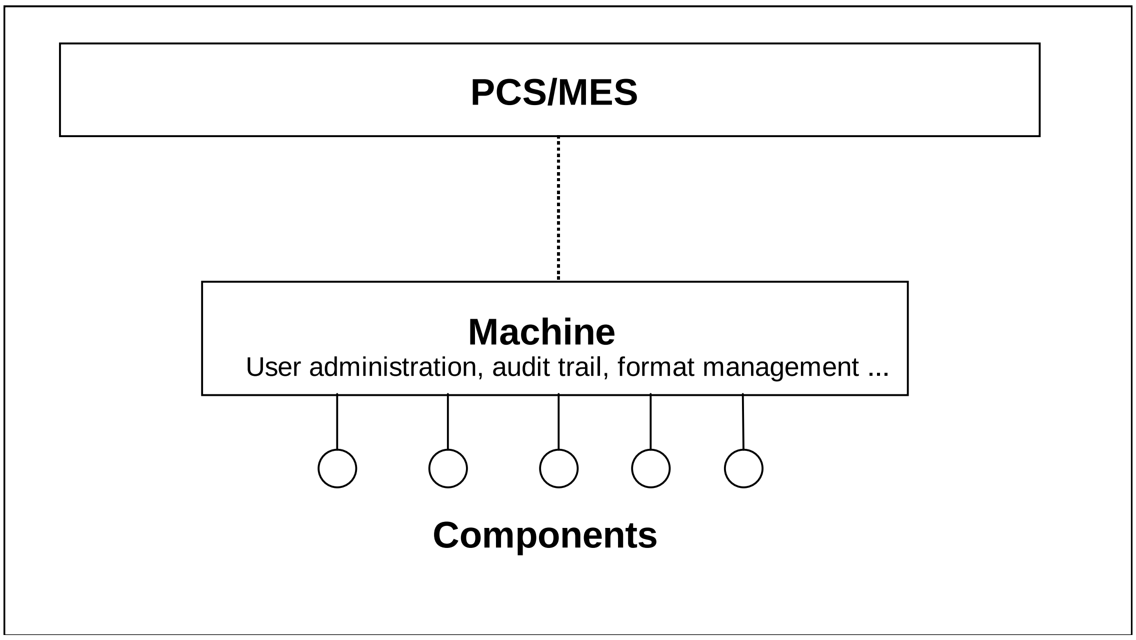
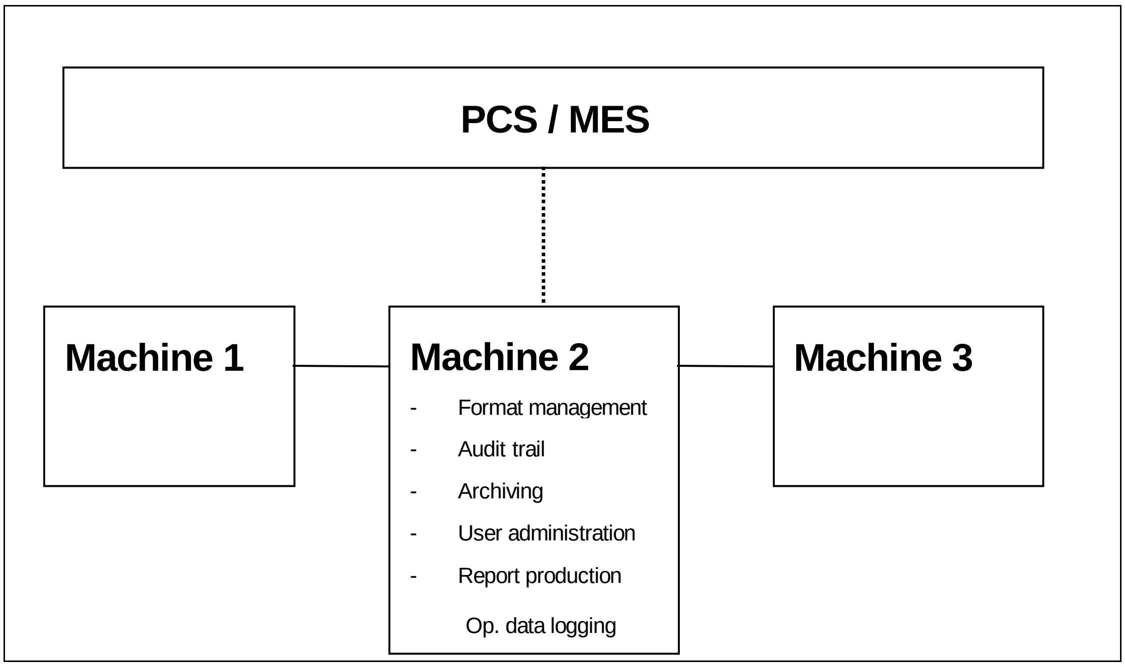


Fig. 2: Scheme for integrating machines and components in production or packaging lines



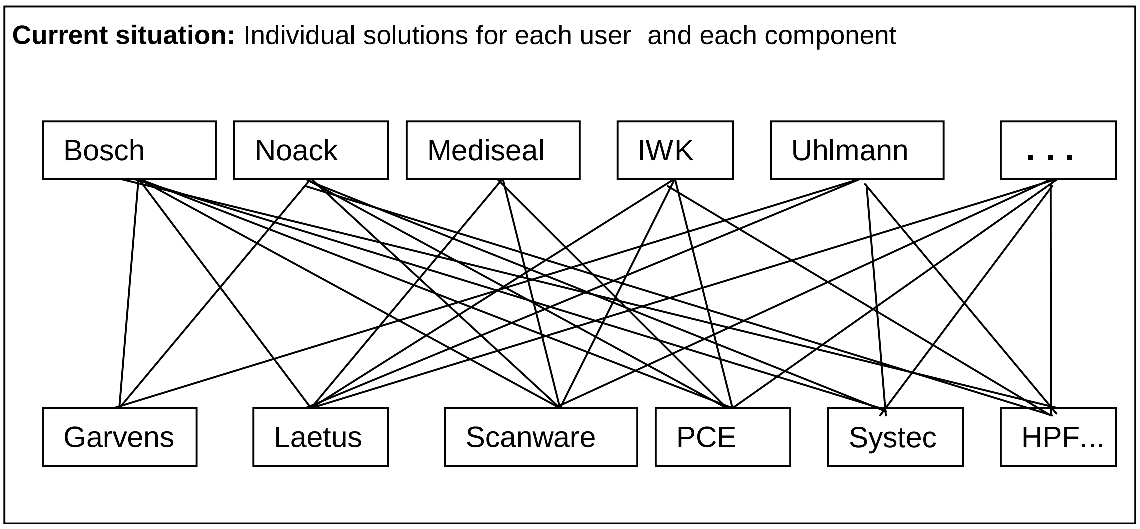
This working paper deals with the integration of intelligent components in a packaging or processing machine (Fig. 1). It should, however, be pointed out here that the considerations set out in this working paper about the data interchange between components and a machine apply correspondingly to the communication between machines of one line or the communication between a machine and its higher-order PCS/MES.

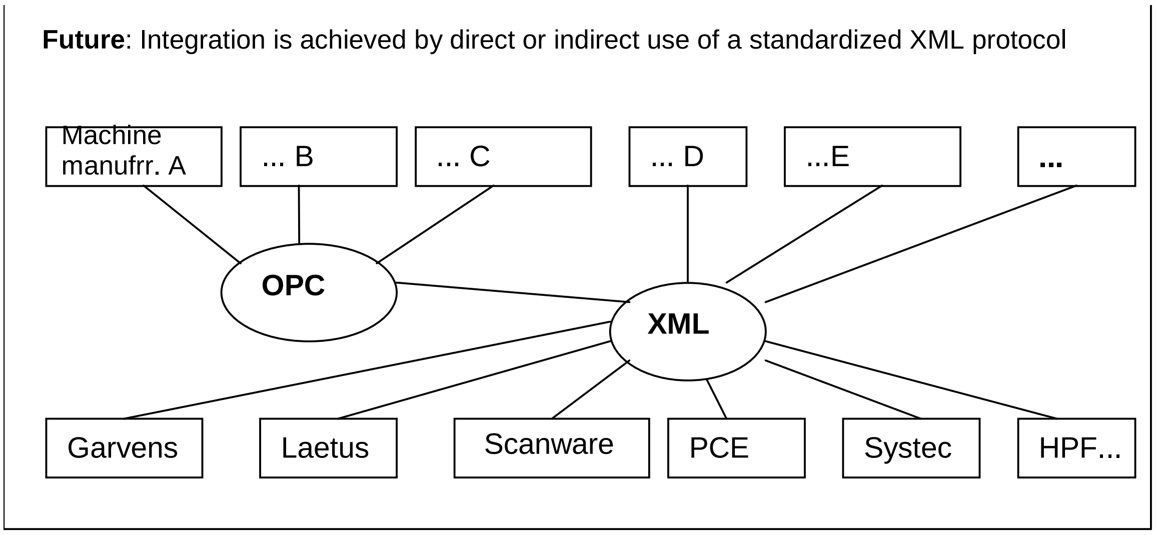
The concept presented here for integrating components in a machine has the objective of locating all responsibilities relating to electronic records in the higher-order machine (Section 3). This approach has the substantial advantage that the machine manufacturer can achieve compliance with the requirements of 21 CFR 11 regardless of the configuration chosen at component level by the user. The consequences of this concept for the data interchange between component and machine are set out in Section 5 by means of examples of a simple and a complex component.

The efforts of integrating intelligent components can be further reduced when the data interchange between the component and the machine necessary to ensure compliance with the requirements of 21 CFR 11 is implemented on the basis of a standardized protocol. This is illustrated by Fig. 3 in which OPC represents communication via intermediate communication layers. In order to reach this objective, an XML protocol called VDMAXML_P is specified in Section 6 which fulfills the requirements set out in Section 3 for data interchange between component and machine.

Communication based on VDMAXML_P makes it easier to meet the requirements for a central administration of user authorizations, a central audit trail or a central format management. For the purposes of supporting the implementation of this requirement as an optional implementation of the protocol, this publication defines the structure of the data interchange for various services such as log-on/log-off, change of password, format activation, audit trail and event log as well as the variables needed for this (Appendix III (informative)). By this means the planning of these services is significantly simplified.

Fig. 3: Reduction of complexity by using standardized protocols


Current situation: Individual solutions for each user and each component
Bosch
Noack
Mediseal
IWK
Uhlmann
. . .
Garvens
Laetus
Scanware
PCE
Systec
HPF...


Future: Integration is achieved by direct or indirect use of a standardized XML protocol
Machine manufrr. A
... B
... C
... D
...E
...
OPC
XML
Garvens
Laetus
Scanware
PCE
Systec
HPF...

The VDMAXML_P protocol is designed for communication between components and a software package — not further defined and in the following called "*communicating party*" — responsible for communication with the other software modules of the machine such as visualization, format data storage, etc. In doing so, the component is not required to know which component of the machine's is the sender or the recipient, respectively, of the relevant message. Such communication structure is advantageous when it comes to the implementation of project-specific requirements, the subsequent integration of components, the implementation of central services as well as the implementation of line control panels.

VDMAXML_P supports the implementation of a communication architecture based on software components acting as message broker. The presence of such a broker architecture, however, is not a pre-requisite for the integration of components by means of VDMAXML_P. Appendix V (informative) contains directions to be followed when implementing a broker architecture in accordance with the VDMAXML_P concept.

The mechanisms specified in the VDMAXML_P protocol can also be used for communicating with a higher-order control system. For this purpose, however, a separate specification is necessary for data interchange between the machine and the control system level.

2 Definition of terms

Term	Definition
Audit trail	In audit trail files, events resulting in the production, amendment or deletion of an electronic record are captured.
Communicating party	Non-specified recipient or sender of VDMAXML_P messages.
DataClient	A component (e.g. visualization) requiring the current contents of a PV. Sender or addressee of VDMAXML_P messages.
DataServer	A component (e.g. a device or control unit) which contributes to the variables housekeeping of the machine. Sender or addressee of VDMAXML_P messages.
Events	Changes in state of process variables or defined events which are to be logged.
GxP	All types of "good practices", e.g. Good Manufacturing Practice.
HMI	All types of user interfaces or operating elements (Human-Machine Interface).
Intelligent component	Functional unit in a machine equipped with independent computing power and its own functional software (including functions for self-monitoring and operational checks).
Manufacturer	The manufacturer of the entire machine.
MES	Management Execution System.
OPC	Open Process Control - a communications concept for the Microsoft Windows environment.
PCS	Process Control System.
Process variable (PV)	Variable used for process control and of importance to several subsystems of the whole system of the packaging or processing machine.
Supplier	The manufacturer of an intelligent component.
TCP client	A client as defined in a TCP/IP network.
TCP server	A server as defined in a TCP/IP network.
VDMAXML_P message	Message in the VDMAXML_P format.
XML	Extensible Markup Language.

3 Requirements arising from 21 CFR 11

When intelligent components are integrated in machines, the responsibilities for the functions falling under 21 CFR 11 lie with the higher-order machine (Overview 1). Accordingly, there is usually no need to document compliance of the integrated component with the requirements of 21 CFR 11. For the higher-order machine, however, the following requirements have to be fulfilled in order to establish compliance of the machine with 21 CFR 11.

- It has to be ensured and documented that the desired values for the process variables of the integrated components match the formats or formulations laid down in specifications.
- The higher-order machine must be able to transfer the desired values of process variables for control of the component to the latter.
- The higher-order machine must be able to verify at any time that the desired values of the process variables of the component in question match the specifications for the current format or formulation (format or formulation comparison).
- If the desired values of the process variables can be changed directly at the component, via an independent HMI, it has to be ensured that these changes can only be carried out by authorized users. To do so, user administration ensues centrally via the higher-order machine (even when logging on via the local HMI). These changes must be reported to the higher-order machine.

- There is no permanent storage of data relevant to quality in the component (e.g. batch protocols).
- The component does not have its own audit trail.

Overview 1: Schematic listing of responsibilities

	Component	Machine[3)]
User administration	-	x
Log-on	(x)[1)]	x
Formulation and format control	-	x
Audit trail	-	x
Reporting changes of specified desired values relevant to quality	(x)[2)]	x
Permanent documentation of process data (archive)	-	x
Report generator		x

1) If the log-on provides authorization to amend desired values relevant to quality, this has to be done by using the machine's central user administration system.
2) If specified desired values can be amended via a local HMI, it has to be ensured that this can only be done by authorized persons and that the change is notified to the higher-order machine.
3) The details of data interchange between component and machine presented in this working paper also apply when the responsibilities listed are transferred from the machine to the PCS/MES level.

4 General technical requirements for data interchange between component and machine

- Lowest possible hardware requirements
- Simple interface
- Simple protocol
- Extensible protocol
- Low network load
- Data transmission in clear text
- Safe transmission and ability to monitor communication
- Fast transmission of small data volumes
- Low coupling
- Low complexity
- Open, manufacturer-neutral solution independent of operating system
- Based on known industry standards
- Scalability to different degrees of complexity of components
- Low-cost integration

5 Data interchange between machine and intelligent components

5.1 Degree of complexity of a component

Features of the component	applies to components of low complexity	applies to components of high complexity
Log-on on the component	no	yes
High number of process variables specific to component	no	yes
Learning system with format management	no	yes
External storage of data in archive	no	yes
Reporting of GxP-relevant events	no	yes
Own audit trail	no	possible; not permitted when integrated in machine
Own report generator	no	possible; not permitted when integrated in machine

5.2 Data interchange between machine and a complex component by means of the example of a camera system

A camera system is characterized in that it is a learning sensor whose reference parameters consist of values which are not definable in advance. To carry out its assigned function, the system is usually equipped with its own user interface. Another characteristic is that numerous process variables of the system are format-dependent. From the GxP perspective, what are the consequences of these characteristics of complex systems for the data interchange between component and machine?

Log-on:
The log-on for using the user interface of the component ensues on the component itself. This also applies to the identification of the user. Once identification has been entered, the user is logged on to the central user administration in the higher-order central user administration system which reports back with the user authorizations entered for the user.

Loading of format-dependent desired values of process variables:
Due to the high number of format-dependent desired values of process variables these are managed on a device-internal basis. The basic settings applicable to the individual formats are documented as part of the qualification. For GxP purposes it is sufficient that the higher-order machine transfers the name and current version of the format to be loaded and that the component reports back that the specified format matches the loaded format. If changes are made to the basic settings during operation, these are to be communicated to the *communicating party*.

Changes to basic setting values (format-neutral basic settings) in the course of configuration management:
Changes in basic setting values are documented within the framework of the change control process during operation and are, accordingly, not part of the audit trail of the machine.
For example, if a force transducer based on a strain gauge is replaced by a piezoelectric one, the desired values of the format are unaffected by this. At the same time, however, the setting parameters of the software may be affected. This, together with the activities for modifying the plant, is documented and validated in the course of ChangeControl.

Archiving:
There is no permanent storage of the GxP-relevant files and data in the component. Insofar as data or files produced by the component are to be archived, these have to be transferred — in a suitable form — to the central archive server. Transfer is documented via the audit trail.

Audit trail:
Responsibility for the audit trail rests with the higher-order machine. The component reports GxP-relevant events. No audit trail is kept by the component.

Report generator:
The component does not draw up any GxP-relevant reports. Data and files which the higher-order machine needs for producing reports are provided by the component on request.

5.3 Data interchange between machine and components of low complexity by means of the example of a temperature sensor

By comparison with a camera system, a temperature sensor is a simple system. It measures the temperature and reports this actual value to the higher-order machine. GxP-compliant production requires calibration of the temperature sensor.

Log-on:
No log-on capability is provided for on the component.

Loading of format-dependent desired values of process variables:
GxP-relevant desired values and tolerances of process variables are not managed on a device-internal basis but rather transferred directly from the higher-order machine to the component. The higher-order machine must be able at any time to verify the set desired values of the GxP-relevant process variables of the component.

Archiving:
An archiving function is not provided for.

Audit trail:
No audit trail is kept by the component. The component reports the actual value of the measured value to the higher-order machine.

Report generator:
It is not possible for the component to produce GxP-relevant reports.

6 Standardized communication between machine and components

Communication between the machine and its components has various levels which are to be described with reference to the ISO/OSI layer model.

Layer	Name	Implemented by	Significance
7c	Application	Programs and con-ventions	Services such as log-on, BatchControl, FormatManagement, etc.
7b		Special process variables	Definition of special process variables and message contents
7a		Process variables	Exchange of process variables
6	Presentation	VDMAXML_P message	Conventions about message structure
5b	Session	Broker, DataClients, DataServers	Agreements about how the broker, the DataClients and the data sources process messages
5a		Sockets	Message transmission
4	Transport	TCP	
3	Network	IP	
2	Data link	Ethernet	
1	Physical		

The assignment of layers 5 and 6 is substantially used for structuring. Layer 5 comprises the actual message transmission via TCP sockets (5a) and the agreed behavior of the participating *communicating parties* (5b).

6.1 Communication layers 1 to 4

For layers 1-4, a TCP/IP network having Ethernet architecture is laid down. Components which do not have a TCP/IP-compliant Ethernet interface and/or have not implemented the VDMAXML_P protocol, must be integrated indirectly via a DataServer.

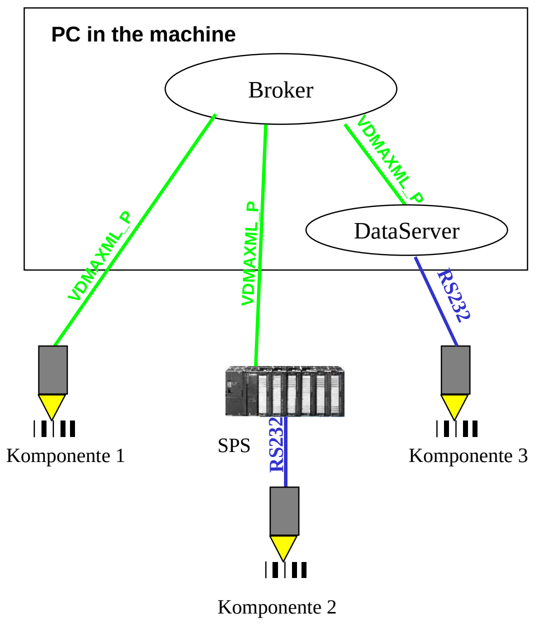


Fig. 5: Indirect integration of devices which are not compliant with VDMAXML_P
(Text in Figure: Komponente=Component)

Fig. 5 presents two methods for integrating a device having only one RS232 interface: Component 2 is connected to the RS232 interface (or I/Os) of a stored-program control (SPC) device. The data of component 2 are present as a copy in SPC variables which are accessed via VDMAXML_P. The SPC sends or receives the data for the component via the RS232 interface and thus plays a proxy role. In the second variant the RS232 interface of component 3 is connected to the PC on which a software package called a DataServer is running which implements the VDMAXML_P protocol to the *communicating party* — e.g. a message broker — and transfers the commands to the RS232 interface. There may be straightforward conversion to a different transmission architecture (RS232) or conversion of contents to a proprietary protocol of component 3.

6.2 Communication layer 5a

Transmission ensues via simple TCP sockets. The DataServer and the devices provide TCP server sockets which the broker can address.
The broker provides TCP server sockets which can address DataClients such as visualization or other programs.
A component is identified by its IP address and its TCP server port.

6.3 Communication layer 5b

This layer lays down how the communicating parties are to behave.

- Each TCP client is responsible for its connections. If a connection is broken off, the TCP client must try regularly to reestablish it.
- Each TCP server ensures that it is addressable.
- Each TCP server is responsible for subordinate devices and connections.
- In the case of implementing a broker architecture, the broker knows all process variables and takes over the routing of the message. It ensures that all subscribers and all other registered data sinks are informed of changes of state of a variable (see also Appendix V).

6.4 Communication layer 6: Standardized XML protocol for communication between machine and component

In the following, a message-based communications strategy for data interchange between component and machine is set out. The addressee of all messages is a software component of the machine, in the following called *communicating party* for simplification. This software component can be a message broker which determines the final recipients of messages and passes the messages on to these. The function of the broker within this communications concept is explained in more detail in Appendix V.

The sender of the message need not know which functional units in the machine finally process the dispatched message. The component communicates with the machine by means of the standardized XML protocol 'VDMAXML_P'.

6.4.1 Structure of VDMAXML_P messages

All VDMAXML_P messages are documents complying with the XML conventions. Moreover, the following agreements apply:

1. Each document consists of a root element <VDMAXML_P> and an element inside that, comprising the actual message content.
2. The single subelement in the VDMAXML_P element has the name of the function of the message (e.g. STATE, PUT).
3. The information needed (apart from the data) is transported in the form of attributes.
4. The data should be short (no large fields or texts). The total length of the string should not normally exceed 2,000 bytes[1].
5. Documents must be transmitted in UTF-8 encoding.
6. The content of an element consists either of a character string or one or more elements. In the latter case blank characters, tab characters or line feeds (CR and/or LF) are permitted for structuring purposes.

6.4.2 Types of VDMAXML_P messages

The subelement of the root element contains the actual message. In the following, the term "message" is generally used as a synonym for this element. The following types of messages are distinguished.

Message type	**Meaning**
STATE	Reports the current new state of a variable
GET	Requests the *communicating party* or DataServer to report the content of a variable
PUT	Requests the *communicating party* or DataServer to set a variable to a certain value
SUBS	Subscribes a variable to the *communicating party* or a DataServer. Whenever the variable changes, the DataServer is to report the current value of the PV. In any case, the DataServer supplies the value of the variable for the first time when the subscribe command is invoked.
UNSUBS	Cancels the subscription of this variable.

Note:
All message types shall be implemented on a component.

[1] The maximum length should be specified by the manufacturer of the machine as a function of the broker used.

6.4.3 *Attributes of a VDMAXML_P message*

In VDMAXML_P messages, only the attributes listed below may be used. Only a few of these need to be understood by all components. If any one of these attributes is not understood by a component it is to be ignored.

Attribute	Name	Meaning	Mandatory	Note
v	variable	Name of variable	X	1

s	storage	Memory category, at present:
		`cur    (current, present value)`
		`mem    (temporarily stored value)`
		`fmt    (format memory)`
		`cmd    (command)`
		`state  (status report)`
		`visu   (reserved for visualization)`
		`vds    (DataServer control)`
		`login  (reserved for log-on inquiries)`
		`msg(message, not stored by communicating party)`
				8

f	from	*ComponentID* of sender		7
t	time	(UNIX) Time stamp		2
k	kind	Kind of data, physical variable, e.g. temperature, pressure		3
u	unit	Physical unit employed, e.g. mm, inch, torr		4
c	command	Defined, embedded commands, at present: i = invalidate		6
d	data type	Data type, e.g.: i1, ui4, number		5

Notes:

1) Names for process variables must be allocated by the manufacturer or agreed with the latter. They, therefore, have global validity.

2) The time stamp is expected and sent in the form of a decimal numerical representation of a 32 bit variable in accordance with the UNIX convention. The time stamp is optional. DataServers should not send a time stamp when the timings of the components are not exactly synchronized (by network timing). If the exact time is relevant, then — in cases of doubt — the timing of the component's *communicating party* is taken as reference.

3) The type of variable is used for consistency testing and can be specified optionally. In general, however, the specification is not used at run time. The internationally agreed variables in the SI system are to be used.

4) The unit of measure used should be stated if it is not certain that it is the same as the system unit of measure for this process variable. If possible the internationally agreed variables of the SI system should be used.

5) The data format relates to the implementation in the device and is independent of the global definition of the process variables. It can be used for consistency testing. Defined data types should be used.

6) These commands offer the possibility of embedding control information in the data stream. Defined so far is:
c="i" (invalidate). In this way a DataServer is to **communicate** that the PV in question is invalid, e.g. because its state could not yet be determined or communication to the device has broken down.
A communicating party shall also be able to process an empty attribut "c" without the communication breaking down.

7) The optional ComponentID should be set when identification of the sender is relevant to a service (e.g. log-on request).

8) Memory class 's' is particularly intended for a broker architecture and has an effect on message routing inside the machine.

6.4.4 Example of a VDMAXML_P message

In the following, the structure of the message is illustrated, by means of which a component indicates that the process variable "FillLevel" has changed to the value "305 (mm)". This is done by means of a "STATE" (status) report. All messages are structured according to the same pattern. Only the "STATE" tag is replaced as required by the meaning of the message, e.g. PUT or GET.

Example

```
<VDMAXML_P>
<STATE t="99458543" s="cur" v="FillLevel" u="mm" >305</STATE>
</VDMAXML_P>
```

Meaning

`<VDMAXML_P>`	*the envelope element of a VDMAXML_P message*
`<STATE`	*the opening tag of a STATE report*
`t="99458543"`	*time stamp*
`s="cur"`	*memory category "cur" = current = present measured value*
`v="FillLevel"`	*name of process variable*
`u="mm"`	*measured value expressed in mm*
`>305<`	*filling level at time of notification = 305 mm*
`/STATE>`	*the closing tag of the report*
`</VDMAXML_P>`	*the closing tag of a VDMAXML_P message*

The message may contain blanks, tab characters and line feeds (CR and/or LF).

6.4.5 Role of DataClients and DataServers in the system

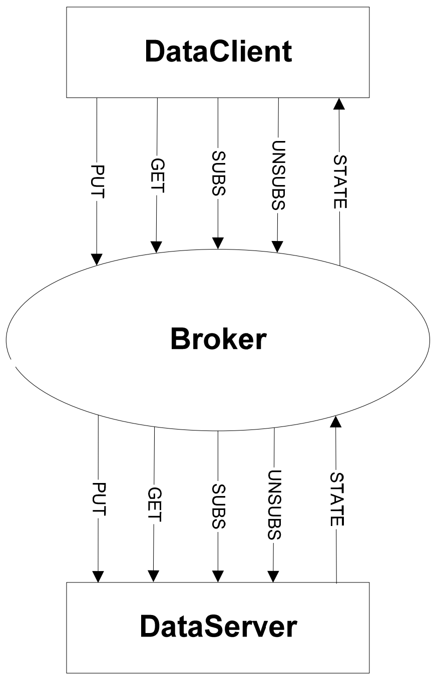
Whether a component plays the role of a DataClient or a DataServer in the system depends on the functionality provided or used at the time by the component.

If the component provides process variables or services of layer 7c, it is a DataServer.

If the component has an interest in variables of another component or if it uses one of the services of layer 7c, it is a DataClient.

If a component makes available its own process variables and itself needs process variables from other components or would like to address services of layer 7c, it must simultaneously be both a DataServer and a DataClient. That is to say the component must implement both the DataClient and the DataServer functionalities specified below.

By means of this binding definition it is laid down which functionality a component must implement in order to successfully communicate with the other components in the integrated system.



The direction of the arrow identifies the flow of the message

Fig. 6: Communication between DataClient and DataServer via a broker

6.4.5.1 DataClient

A DataClient may send only the following messages set out in Section 6.4.2:

- GET
- PUT
- SUBS
- UNSUBS

A DataClient may only process STATE messages. All other messages are not permissible. A DataClient must not respond to an invalid message but it must ensure that it continues to function correctly. It should log the invalid message. It processes only STATE messages which it either receives in response to a sent GET or receives at undefined intervals from the *communicating party* due to a previously executed subscription (SUBS).

A DataClient must not send any STATE messages, as these always indicate just the state of a process variable and a client does not possess its own process variables.

A DataClient is the initiator of a communication process. It sends GET, PUT, SUBS or UNSUBS messages to the *communicating party* which then passes these on to the corresponding DataServer or service provider. The latter in turn sends a reply in the form of a STATE message for a GET and a SUBS. In the case of a SUBS message, the DataClient then receives for every change in the subscribed variables a corresponding STATE message which has been initiated by the DataServer. If no answer is received within a defined time (timeout) it can assume that there is an error in the system.

Following a PUT message, the client receives no direct answer back. The DataServer receiving the message will attempt to set the process variable and then send a STATE report to the system, containing either the newly set process value or "invalidate" if for some reason the value could not be set. In order that the client which initiated the PUT is informed of this change or can check the PUT, there are in principle two possibilities:

- it sends an explicit GET for this variable in order to interrogate the value once again

or

- prior to the PUT it has registered itself by a SUBS for changes in the values of these process variables.

The DataClient receives no reply to an UNSUBS message. Afterwards it receives no further STATE reports for this variable. Due to the undefined timing behavior, however, it is certainly possible for the DataClient after sending an UNSUBS message to continue to receive STATE reports if these have already arrived at the *communicating party* before the UNSUBS was sent but are only distributed after this. The DataClient must be prepared for this and must not respond with an error.

The client can indicate an error situation to the user but it must ensure to continue functioning correctly.

If a component is additionally to provide DataServer functionality, it must also implement DataServer functionality (see Section 6.4.5.2).

6.4.5.2 DataServer

A DataServer may send out only STATE messages. A DataServer never sends a GET, PUT, SUBS or UNSUBS message. A DataServer sends out STATE messages only for process variables which are identified in it as subscribed or are requested by a GET. A subscription exists when the DataServer receives a SUBS message for a process variable from the *communicating party*.

On the other hand, a DataServer may receive only GET, PUT, SUBS and UNSUBS messages. It must not respond to invalid messages and must ensure to continue functioning correctly.

A DataServer knows no reference count for SUBS and UNSUBS messages. If it receives a SUBS message for a process variable, it marks this as subscribed and thereafter for each change in the process variable sends a STATE message to the *communicating party*.
Other SUBS messages are answered by the DataServer with a STATE message containing the current value of the variable. In the case of an UNSUBS message the variable is again labeled as not subscribed and further UNSUBS messages are ignored by the DataServer.

In the event of a loss of connection to the *communicating party* or a communications error, the DataServer deletes all subscriptions and ensures that it can be reached again from its server port. It then waits for the *communicating party* to establish a connection to it.

If a component is additionally to provide or need DataClient functionality, it must also implement DataClient functionality (see Section 6.4.5.1).

6.4.6 Definition of message flow

The message flow of each message is described below.

It is generally the case that for each nonexecutable command an invalidate in the form of a STATE message is sent from the component which can first find the error. This may be the *communicating party*, or the DataServer for which the command is destined.

In version 2.0 of the specification, a global process variable always exists in precisely one DataServer. There are never two DataServers from which the same global variable is supplied.

6.4.6.1 Message flow in the case of PUT messages

A PUT message is always initiated by the DataClient. The purpose of the message is to set a process variable to a defined value.

1. The client sends the PUT message to the *communicating party*.
2. The *communicating party* identifies the DataServer which provides the process variable and passes the PUT message on to it. The project design work must ensure that only one DataServer is nominated as the source for a process variable.

3. The DataServer receives the message and sets the required process variable to the value specified in the message.
4. If there is a subscription for this variable, it sends a STATE message to the *communicating party*. The STATE message contains the current, altered value of the process variable or "invalidate" if the value could not be set.
   If there is no subscription the message flow ends at this point.
5. The *communicating party* passes the STATE message on to all DataClients which have registered for this process variable by a SUBS message.

6.4.6.2 Message flow in the case of GET messages

A GET message is always initiated by the DataClient. The purpose of the message is to interrogate explicitly the current value of a process variable.

1. The client sends the GET message to the *communicating party*.
2. The *communicating party* identifies the DataServer providing the process variable and passes the GET message on to this server.
3. The DataServer receives the message, reads off the current value of this variable and sends a STATE message to the *communicating party*. The STATE message contains the current value of the process variable or "invalidate“ in the event that the value of the process variable is invalid at the time or has not been set.
4. The *communicating party* passes the message on to the DataClient that initiated the GET message and to all DataClients which have registered for this process variable by a SUBS message, even when the value of the process variable has not changed.

6.4.6.3 Message flow in the case of SUBS messages

A SUBS message is always initiated by the DataClient. The purpose of the message is to log on for a change in the process variable.

1. The client sends the SUBS message to the *communicating party*.
2. The *communicating party* identifies the DataServer providing the process variable and passes the SUBS message on to this server. Additionally, it registers the DataClient internally as a subscriber to this variable. All future STATE messages received by the *communicating party* for this variable it passes on to this DataClient until it receives a corresponding UNSUBS message from the client.
3. The DataServer receives the message and internally registers this variable as subscribed. For each future change in this variable the DataServer initiates a STATE message and sends this to the *communicating party*.
4. In response to the SUBS message the DataServer sends an initial STATE message to the *communicating party*. The STATE message contains the current value of the process variable or "invalidate“ in the event that the value of the process variable is invalid at the time or has not been set.
5. The *communicating party* passes the STATE message on to all DataClients which have registered for this process variable by a SUBS message.
6. Further SUBS messages for this variable are answered by the server with a STATE message.

6.4.6.4 Message flow in the case of UNSUBS messages

An UNSUBS message is always initiated by the DataClient. The purpose of the message is to notify withdrawal from a registered interest in change in a process variable which had been notified by a SUBS message.

1. The client sends the UNSUBS message to the *communicating party*.
2. The *communicating party* removes its internal entry identifying the client's subscription to this process variable. Only in the event that this is the last (single) DataClient which has subscribed to this process variable does the *communicating party* pass on the UNSUBS message to the corresponding DataServer.
3. The DataServer receives the UNSUBS message and removes the variable from its internal list of subscribed variables. After this it sends no further STATE messages regarding changes occurring in this process variable. Nor does it respond to the UNSUBS message.

6.4.6.5 Message flow in the case of STATE messages

A STATE message is always initiated by the DataServer. The purpose of the message is to indicate a change in a process variable. When a change occurs in the value of a variable, the DataServer sends STATE messages only when there are subscriptions to this variable. Changes in process variables for which there are no subscriptions, are not indicated by STATE messages.

1. The DataServer sends the STATE message to the *communicating party*.
2. The *communicating party* passes the message on to all DataClients which have registered for this process variable by a SUBS.
3. Following the STATE message the DataServer receives no reply message from the *communicating party*; likewise no reply messages are generated by the DataClients which receive a STATE message from the *communicating party*.

6.5 Communication layer 7a: Interchange of process variables

Process variables are variables which are used to control the process and are important for several subsystems of the full system machine. They not only represent process parameters but are also used to communicate commands, status changes, etc. (see Section 6.6). They are, therefore, subject to conventions, e.g. regarding naming. Accordingly, a description must first of all be drawn up for each process variable (PV) introduced.

6.5.1 ComponentID

Normally, the status of process variables in the entire system is reported to all components which have subscribed to these variables. For various services (e.g. log-on or status inquiries), however, it is unavoidable to address a device or another component directly. Accordingly, each component has to be given a unique and configurable name, its ComponentID. This ComponentID shall only consist of the characters 0–9, a–z and A-Z.
All variable names intended to be applicable to only one component begin with the ComponentID and an underline character.

6.5.2 Associated data

In version 2.0 of this specification, access to process variables usually ensues on the basis of single variables, regardless of how these are organized technically as data in the component.
In certain cases this approach is not appropriate because a PV only makes sense in association with other PVs and the consistency of the association is not ensured in the case of independent access. This is the case, for example, when instantaneous values are involved which are no longer valid at the time of the next access, if the sequence of accesses is important or when a command is to be transmitted with the associated command parameters. In this case associated data are used which are transmitted in the form of XML elements containing items of structure information.

It is explicitly pointed out that associated data are intended only for the purpose set out above. The use of associated data for structuring purposes is not permissible.

The following rules serve as a basis for the implementation of "associated data".

1. The content of a message type may consist only of
   (a) a single character string (xs:string) or
   (b) an XML element possibly containing other subelements (referred to as the structure element for associated data).

2. The character string <VDMAXML_P> must not be used as part of the content of the message type since it is used for synchronization of communication.

3. A structure element can in turn have the following contents:
   (a) A single character string in which the data type can be more precisely defined by the attribute "d" in the surrounding XML starting element but does not have to be (e.g. d="i2"). If no attribute 'd' is transmitted, the data type is interpreted as 'string'.
   (b) XML elements with the same element name represent a complete array (see "UserRoles" in the process variable "*ComponentID*_UserPerm"). The sequence of the XML elements determines the index of the array element in question.

   (c) Different XML elements are the members of the substructure and are subject to the rules of a structure element.

4. XML elements for representing "associated data" must not possess any attributes. The sole exception is attribute "d" for the structure element whose content is a single character string (see 3a).

5. In the case of arrays, all elements must always be transmitted.

6. A structure element of one type always contains the same subelements.

7. The messages GET, PUT, SUBS and UNSUBS can only be executed for the whole structure and **not** for individual elements. A data server **always** provides the whole structure in a STATE message.

Examples:

A) Two-dimensional specification of a position for a sensor:

```
<VDMAXML_P>
     <STATE v="Sensor" s="cur">
          <Position>
               <X>7</X>
               <Y>13</Y>
          </Position>
     </STATE>
</VDMAXML_P>
```

B) Process variable "*ComponentID*_UserPerm" for component “Panel1”:

```
<VDMAXML_P>
     <STATE v="Panel1_UserPerm" s="cur">
          <UserPerm>
               <UserLoginName>ADMIN</UserLoginName>
               <UserID>ADMIN</UserID>
               <UserFullName>Global
SystemAdministrator</UserFullName>
               <LoginResult>PasswordInvalid</LoginResult>
               <UserRoles>
                    <Role>Technician</Role>
                    <Role>Vision</Role>
               </UserRoles>
               <Expires>31</Expires>
          </UserPerm>
     </STATE>
</VDMAXML_P>
```

6.6 Communication layer 7b: Definition of special process variables and message contents

In addition to process parameters, commands and status changes can also be communicated by means of VDMAXML_P. For this purpose, the special memory categories specified in attribute "s“ as well as special process variables for the control of the DataServers are used. These special process variables shall be subject to special agreement, if applicable.

6.6.1 Special memory categories for the transmission of commands and status changes — attribute 's'

The attribute "s" (memory class) of a VDMAXML_P message determines the routing. Thus, a distinction can be made in the data flow between, for example, genuine process variables and commands. In particular this determines whether a message is to be stored by the component's *communicating party* and whether it is passed on to DataServers.

Name	**Meaning**
cmd	Command, e.g. from visualization to another DataClient. These messages are not to be stored in the system (event message)
cur	Instantaneous values from the process, setting values
fmt	PV as stored in format
login	Messages sent exclusively to the log-on server (event message)
mem	PV when it is temporarily stored externally
msg	Message; not stored by the *communicating party*
state	Status messages, e.g. when a command has been executed
vds	Control of DataServers
visu	Private information of the visualization systems

Event messages are messages that are valid only at the moment of origin. In a broker architecture, the message broker distributes them just once to the current subscribers. After that the information they contain cannot be interrogated any longer. Accordingly, a client which subscribes to the corresponding PV only after it has been distributed receives only the next transmission. All other PVs can be interrogated at any time as to their current contents.

6.6.2 Internal variables for the control of the DataServers and for diagnostic purposes

The following variable names are reserved for the control of the DataServer and for diagnostic purposes:
*ComponentID*_StartServer
*ComponentID*_StartScan
*ComponentID*_ServerStatus
*ComponentID*_ServerMessage.

The content of these variables will be specified in a later version of this protocol.

The memory class for internal variables is ***vds***; by this means the component's *communicating party* can ensure that the sending component has the authorization to control the DataServers. The global variable name is composed of the ComponentID of the DataServer and the variable designation identified above, e.g.:
"vds ComponentID_StartServer".
If a DataServer cannot operate these internal variables, it must not be influenced by accesses thereto.
DataServers can supply any other internal variables as required, e.g. error counters or extended status displays.

Example

<PUT s="vds" v="*ComponentID*_StartServer">1</PUT>.

6.6.3 Special process variables for the implementation of central services

The implementation of central services requires special process variables to be agreed upon. In Appendix III (informative) to this publication, examples are given for the implementation of the following services:

- Log-on/log-off service
- Change of password service
- Format service
- Audit trail service
- Event log service

The special process variables specified there do not belong to the normative part of the specification of the communication protocol VDMAXML_P. They may be subject to special agreement.

6.7 Communication layer 7c: Services

The communication protocol VDMAXML_P can be used to implement services supporting central user administration, a central audit trail, a central format management, etc. The implementation of such services requires special process variables to be agreed upon.

In Appendix III (informative) to this publication, examples are given for the implementation of such services, based on the communication protocol VDMAXML_P. The following services are specified:

- Log-on/log-off service
- Change of password service
- Format service
- Audit trail service
- Event log service

The specification of such services does not belong to the normative part of the specification of VDMAXML_P. The fact that a component supports the communication protocol VDMAXML_P, does not imply that the component supports also the services specified in Appendix III. This is subject to special agreement. Thus, in the case of components supporting the protocol VDMAXML_P, it is recommended to mention additionally which of the services specified in Appendix III are supported.

Appendix I (normative)
Data types used in VDMA_XML messages

This section shows the key data types used in VDMAXML_P messages. Not all data types need actually to be implemented. The following data types are based for the most part on the XML data specification.
(http://www.w3.org/TR/1998/NOTE-XML-data)

Name	**Parse type**	**Memory type**	**Examples**
number	A number with as many digits as required, optionally with decimal places, optionally with preceding sign, optionally with exponent. Punctuation in line with US English	string	15 3.1415 123.456E+10
int	A whole number with or without preceding sign	32 bit integer	1 -12345
float	As for *number*	64-bit IEEE 488	0.31415 79E+1
fixed.14.4	As for *number* but with no more than 14 digits before and 4 digits after the decimal point	64-bit integer	12.345
boolean	"1" or "0"	bit	
dateTime.iso8601	Date in ISO-8601 format. Optional time, fractions of a second can be expressed in nanoseconds.		2004-03-08T08:15Z (Note 1)
dateTime.iso8601tz	Date in ISO-8601 format. Optional time, fractions of a second can be expressed in nanoseconds.		2004-03-08T08:15:05Z (Note 1)
date.iso8601	Date in ISO-8601 format. Without time.		2004-03-08 (Note 1)
time.iso8601	Time in ISO-8601 format. Without date.		08:15:05Z (Note 1)
i1	Whole number, optionally with preceding sign	8 bit integer	0, -128, +23, 45
i2	Whole number, optionally with preceding sign	16 bit integer	12, +465, -32768
i4	Whole number, optionally with preceding sign	32 bit integer	
i8	Whole number, optionally with preceding sign	64 bit integer	
ui1	Whole number without preceding sign	8 bit unsigned	
ui2	Whole number without preceding sign	16 bit unsigned	
ui4	Whole number without preceding	32 bit unsigned	

	sign		
ui8	Whole number without preceding sign	64 bit unsigned	
r4	As for *number*	IEEE 488 4 bytes float	
r8	As for *number*	IEEE 488 8 bytes float	
float.IEEE.754.32	As for *number*	IEEE 754 4 bytes float	
float.IEEE.754.64	As for *number*	IEEE 754 8 bytes float	
uuid	Hexadecimal numbers representing octets; optionally embedded hyphens are to be ignored.	128-bytes Unix UUID structure	F04DA480-65B9-11d1-A29F-00AA00C14882
uri	Universal resource identifier	According to W3C specification	
bin.hex	Hexadecimal numbers representing octets	No specified size	
char	String	1 character (UTF-8)	
char.ansi	String	1 ASCII character	
string	Pcdata	String (UTF-8)	αβχδεϕγ
string.ansi	String containing only ASCII characters (<=255)	(8 or 16 bits per character)	This is a simple text.
string.ansi.long	Very long ASCII text		This is a text possibly several Mbytes in length....
string.long	Very long Unicode text	String (UTF-8)	This is a text possibly several Mbytes in length....
timestamp	Time stamp		
timestamp.unix	Time stamp according to Unix convention	32 bit Integer	978618766 (= 4. Jan 2001, 15:32:46)

No connected device or DataServer must be caused to break down by an unknown data type. In cases of doubt, an error message can be output or recorded in a log file.

Notes:
1) Hours, minutes and seconds in time information shall always be stated in double digits, with a leading zero if necessary. Hours are expressed in the 24 hour system. All time information relates to the current time zone. If times independent of time zones are to be given, the letter "Z" shall be appended to the time information and the time must be converted to UTC (Universal Time Coordinated).

Appendix II (normative)
Text conventions

Character encoding
For communication via VDMAXML_P, character encoding in accordance with the UTF-8 standard is laid down. This encoding system functions reliably on all file systems and is compatible with all previous systems, provided only the US-ASCII character set (decimal 1 to 127) is used. Characters having a code >127 are used for encoding all other UNICODE characters having different numbers of bytes/characters. Program files, configuration data, etc. are expected as a rule in the form of ASCII files.

Note:
UTF-8 and UNICODE are conventions for encoding characters. The question as to whether a character can actually be represented, however, depends on whether an appropriate program is used and an appropriate font is available. In cases of doubt the required character sets must be agreed between final customer, manufacturer and supplier.

XML documents
Communication between the DataServer of the component and its *communicating party* on the higher-order machine takes place by means of XML documents. These are readable texts which must follow certain conventions so that they can be transmitted without problem. The following conventions relate to different types of texts or documents. The structure of the documents is set down in XML pro forma (W3C). In different components (tag names, attributes, contents), XML documents must in each case contain only a certain number of characters. For the XML protocol for data interchange between component and machine specified here, the XML specifications were further restricted in order to simplify processing of the XML documents. The restrictions set out below are valid even if the XML specification allows otherwise.

Table AII.1: VDMAXML_P: additional restrictions for XML documents

Where	**Restrictions**
Tag name	Must not start with "xml" (upper and lower case letters). 1st character must be a US-ASCII letter 2nd to nth characters: US-ASCII letters, numerals, underline character
Attribute name	1st character must be a US-ASCII letter 2nd to nth characters: US-ASCII letters, numerals, underline character, full stop, comma
Attribute contents	Contents containing an **"**, must be enclosed in **'**. Contents containing an **'**, must be enclosed in **"**. Instead of **'** and **"** in attribute contents, the character entities **&apos;** and **&quot;** may be used. The characters **&**, **<** and **>** must be replaced by their character entities **&amp;**, **&lt;** and **&gt;**. Line feed, carriage return and tabs must be represented by their numeric character references (**&#10; &#13;** and **&#9;**).

Table AII.2: Names of process variables:

Where	**Restrictions**
Data contents	Names of process variables must not contain any characters prohibited in XML.

Appendix III (informative) Examples of implementing services based on communication protocol VDMAXML_P

All defined services are implemented as a rule in the form of DataServers. In order to address the latter, the addressing component must be a DataClient. If a service is unable to afford its required functionality (e.g. because database links are not available), it quits its server socket and in this way removes itself from the system. This provides an unambiguous signal that the service is not addressable. A service should be implemented in such a way that the downtime is as short as possible and that it can take part in the system again as quickly as possible.

When implementing the services, it is recommended to follow the examples with regard to the sequence of the data transmission for the respective specified variables.

Log-on/log-off service

The log-on service is a DataServer component. It provides the process variables "LoginRequest“ and "LogoutRequest“ which can be used by a system client for logging on or logging off, respectively. The parameters transmitted to the variable "LoginRequest“, are the log-on name, the password (encrypted, if applicable) and the ComponentID of the DataClient on which the log-on takes place. In the case of the process variable "LogoutRequest“, only the ComponentID is necessary. The user currently registered on this client is logged off.
When starting the LoginServer, all UserPerm variables subscribed to shall be set to <LoginResult>Logout</LoginResult>, all roles to empty and Expire to 0.
The log-on component forms part of the system's user administration.

If, on grounds of safety, the password is transmitted only in encrypted form, an asymmetric encryption system should be employed. Components, where log-on is possible, should support at least the RSA encryption system.[2]

As a general rule each component in the system has a ComponentID. For each component implementing its own authorizations structure or having an interest in logging on, the log-on service provides an independent process variable: "*ComponentID*_UserPerm“.
A user can assume several roles simultaneously in the system.

[2] If it is planned to export the machine or component to the United States of America, US import restrictions on encryption systems have to be observed.

Memory class (s)	Name (v)	Data	Meaning
login	LoginRequest	ComponentID UserLoginName, Password	A DataClient logs a user on.
login	LogoutRequest	ComponentID	A DataClient logs off the user currently logged on.
cur	PublicKey[3]		The public key in the RSA 1024 bit encryption system
cur	*ComponentID_* UserPerm	UserLoginName, UserID, UserFullName, LoginResult, UserRoles, Expires	The log-on server reports to the system the authorizations of the user just logged on in the form of a STATE message. By means of the UserRoles element, a role code or several roles assigned to the user or a role profile can be transmitted. The Expires element states the remaining life of the password in days.
cur	*ComponentID_* *UserName*	UserID, UserLoginName, UserFullName	Used to inquire about the user currently logged on.

Log-on sequence:

1. All components with an interest in log-on activities register themselves by a SUBS to obtain their corresponding process variable "ComponentID_UserPerm".
2. A user logs on to a DataClient (e.g. terminal Term1). During this process, the DataClient requests the user's log-on name and password.
3. The DataClient sends a PUT message for the PV "LoginRequest" of the log-on server to the *communicating party*.
4. The *communicating party* passes the message ONLY on to the registered server for log-ons since it belongs to the "login" memory class.
5. The log-on server verifies the user name and the password.
6. The configuration of the log-on server determines which components are assigned to the client carrying out the log-on and which local roles (for the component in question) the logged-on user assumes. From this information a STATE message for the *ComponentID*_UserPerm variables for the component in question is generated for each of these components and sent to the *communicating party*. The messages each contain an associated data set comprising:
   a) LoginResult: Logout, Success, LoginRefused, UnknownUser (optional), PasswordInvalid (optional)
   b) UserRole (name of role) for each role that the user assumes provided log-on was successful
   c) UserFullName provided log-on was successful
   d) UserID provided log-on was successful
   e) Expires: remaining duration of validity of the password in days (0 = expired, -1 = does not expire).
7. The *communicating party* distributes the messages to the corresponding subscribers to the PV "*ComponentID*_UserPerm".
8. The inquiring DataClient must have subscribed to the PV "*ComponentID*_UserPerm" applicable to it. A DataClient may also be configured in such a way that it subscribes (exclusively or additionally) to the UserPerm variable of other DataClients, as a result of which synchronization of access rights even across several terminals and/or devices is possible.

[3] Only required if encryption has been agreed upon between the communicating partners.

Log-off sequence:

1. A user logs off from a client.
2. The client sends the PUT message "LogoutRequest“ to the *communicating party*. This message contains the ComponentID of the client from which the user currently logged on is logged off.
3. For all components assigned to the client the log-on server sets the corresponding process variable "*ComponentID*_UserPerm“ for the component and sends a STATE message containing an associated data set comprising:
   a) LoginResult: log-off
   b) UserRoles blank
   c) UserFullName (optionally blank)
   d) UserID (optionally blank)
   e) Remaining duration of validity of the password: 0
4. The *communicating party* passes the messages on to the components registered for the messages.

Details:
The memory category "login" in the LoginRequest message ensures that the message is passed on by the *communicating party* only once and only to the log-on server. It is not to be stored either by the *communicating party* itself or by other components.
A constituent of the names used *is ComponentID*. This is the configured identification of a DataClient. It is always needed when general messages are destined only for individual components.
The DataClient or the log-on server should send an "event" message which is registered by an event logger. The log-on server is to respond autonomously to false log-on attempts, etc. The attached devices receive only the information identified above.
No special action is needed for log-off since each DataClient can send a log-off message.
In the event of a log-on only the LoginResult=Success message signals a successful log-on. Any other answer means that the log-on attempt has failed. A specific implementation of the log-on server can instead of returning the standard message "LoginRefused“ give more specific details such as "PasswordInvalid“.
The roles and user groups must be configurable. Each DataClient must know the role definitions relevant to it. Since the role designations need to be understood only by the receiving device, they may also contain lists or complex data (e.g. profiles).

Examples of messages:

```
<VDMAXML_P>
        <PUT v="LoginRequest" s="login">
                <LoginRequest>
                        <ComponentID>Panel1</ComponentID>
                        <UserLoginName>ADMIN</UserLoginName>
                        <Password>xyz</Password>
                </LoginRequest>
        </PUT>
</VDMAXML_P>

<VDMAXML_P>
        <PUT v="LogoutRequest" s="login">
                <LogoutRequest>
                        <ComponentID>Panel1</ComponentID>
                /<LogoutRequest>
        </PUT>
</VDMAXML_P>

<VDMAXML_P>
        <STATE v="Panel1_UserPerm" s="cur">
                        <UserPerm>
                                <UserLoginName>ADMIN</UserLoginName>
                        <UserID>ADMIN</UserID>
                        <UserFullName>Global System Administrator</UserFullName>
                        <LoginResult>PasswordInvalid</LoginResult>
                        <UserRoles>
                              <Role>Technician</Role>
                              <Role>Vision</Role>
                        </UserRoles>
                        <Expires>31</Expires>
                  </UserPerm>
      </STATE>
</VDMAXML_P>

<VDMAXML_P>
      <STATE v="Panel1_UserName" s="cur">
            <UserName>
                  <UserLoginName>ADMIN</UserLoginName>
                  <UserID>ADMIN</UserID>
                  <UserFullName>Global System Administrator</UserFullName>
            </UserName>
      </STATE>
</VDMAXML_P>
```

Configuration of the log-on server

The configuration of the log-on server knows an assignment of users to global user groups. A user group is a logical grouping of users having equal authorizations. Users and user groups have a structure analogous to the user management in a Windows system.
In addition, the log-on server knows all DataClients on which a user can actively register by a log-on (input of user name and password via a user interface). Components for which a corresponding log-on is to be carried out can be assigned to any LoginClient when a user logs on to this client. On the basis of the association between the user and the client which the user has logged onto, it is determined for each component assigned to the client what the roles local to the component for this user are. These are sent as a STATE message via the "*ComponentID*_UserPerm" variable of the corresponding component. The component itself is then responsible for allocating its internal authorizations on the basis of the roles transmitted.
Any DataClient which users can log onto is itself responsible for its automatic log-off. If for a configured time no input is made, it sends a corresponding log-off message to the system. All associated components then receive the corresponding message that the user was logged off.

Change of password service

If, on grounds of safety, the password is transmitted only in encrypted form, an asymmetric encryption system should be employed. Components, where log-on is possible, should support at least the RSA encryption system.[4]

Memory class (s)	Name (v)	Data	Meaning
login	**ChangePasswordRequest**	ComponentID UserLoginName Password	The element 'Password' contains the subelements 'Old' and 'New.'
cur	***ComponentID*_ChangePasswordStatus**	ComponentID UserLoginName UserID UserFullName ChangePasswordResult	The variable contains data needed for presenting the result of the request for a password change. For the element ChangePasswordResult, the following states are defined: Success Refused UnknownUser (optional) PasswordInvalid (optional).

Examples of data:

```
<VDMAXML_P>
        <PUT v="ChangePasswordRequest" s="login">
                <ChangePasswordRequest>
                        <ComponentID>Panel1</ComponentID>
                        <UserLoginName>ADMIN</UserLoginName>
                        <Password>
                                <Old>Base64 encoded password</Old>
                                <New>Base64 encoded password</New>
                        </Password>
                </ChangePasswordRequest>
        </PUT>
</VDMAXML_P>
```

```
<VDMAXML_P>
        <STATE v="Panel1_ChangePasswordStatus" s="cur">
                <ChangePasswordStatus>
                        <ComponentID>Panel1</ComponentID>
                        <UserLoginName>ADMIN</UserLoginName>
                        <UserID>ADMIN</UserID>
                        <UserFullName>Global System Administrator</UserFullName>
                        <ChangePasswordResult>Success</ChangePasswordResult>
                </ChangePasswordStatus>
        </STATE>
</VDMAXML_P>
```

[4] If it is planned to export the machine or component to the United States of America, US import restrictions on encryption systems have to be observed.

Format service

The format service is a DataServer component. This provides the process variables defined below which are needed for data interchange for the purpose of format activation and format management.

Memory class (s)	Name (v)	Data	Meaning
cmd	SetFormat	FormatID FormatName (optional) FormatVersion FormatStatus	Format (formulation) specified by central format system. For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'.
cur	CurrentFormat	FormatIDFormatID FormatName FormatVersion	The specified format. If the specified format cannot be correctly loaded the content of each of the subelements is a 'blank string', that is an empty tag.
cmd	SaveFormat	FormatID FormatName FormatVersion FormatStatus	Stores the current process values as format. For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'

cur	Formats	FormatID=0
		FormatName=0
		FormatVersion=0
		FormatStatus
			Returns all known formats in an associated data set.
			For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'

Format management is done at the process variable level, i.e. all process variables identified as relevant to format are set when loading a format, or continuously deposited in a database when storing a format.

Format management knows all process variables to be taken into account in the event of a format change. When there is a change in format, the format management system sends a PUT for each process variable and sets it to the value stored in the corresponding format. It then sends a GET for each process variable and checks whether it was possible to set the value. In the event that it was possible to set ALL of the process variables correctly, the format is taken as correctly set. Even if only one process variable could not be correctly set, the format is taken as not correctly set. This status is shown in turn by the format management through the process variable "CurrentFormat“ in the form of a STATE message.

A component possessing format-relevant process variables, must not delay setting the variables as otherwise the format management cannot test correctly whether the change of format was ordered or not.

A complex component requiring format-dependent configuration data, which it manages internally, provides only one process variable for format activation. If this is correctly set, the format counts as transmitted. The complex component may then still be occupied with the internal format change. The format management system cannot test this more precisely. For this case the component should make available special process variables which a client can check correspondingly. These process variables are an additional property of the component which a client can interrogate via the normal interchange of process variables. These have no direct link with format management.

The format management system should make an AuditTrail entry indicating that a format has been loaded or stored. In order that this can be done correctly, the format management system must register with UserManagement for log-on process variables so that it can learn who the current user is. Furthermore, it must contact the AuditTrail service in order to make the AuditTrail entry.

The AuditTrail entry contains only the information that a corresponding format has been set. The actual process values are not logged since these are filed externally where they can also be logged or viewed.

Procedure for setting a format:

1. A client calls for the setting of a certain format by a PUT on the process variable "SetFormat“ by transferring the unique format number, the format name (optional), the format version and the status.
2. The format management reads all process variable values for the desired format identified as format-relevant, and sends a PUT to the *communicating party* for each of these values. In the case of complex components the specified format can be activated by a PUT on the corresponding process variable.
3. The *communicating party* passes the PUT messages on to the corresponding components.
4. The format management sends a GET for each format-relevant process variable in order to check whether it was possible to set the values correctly. In the case of complex components the check is carried out by means of a GET on the corresponding process variable.
5. The *communicating party* passes the GET messages on to the corresponding components.
6. The components respond with STATE messages and, in those, supply the current values of the process variables.
7. The components' *communicating party* passes the STATE messages on to the format management.
8. The format management checks the STATE message against each PUT message. If ALL the STATE messages match the PUT messages, the format is taken to be successfully set, otherwise not.
9. The format management sends a STATE message for the process variable "CurrentFormat“ to indicate that the format has been correctly set. If the specified format cannot be correctly loaded, the content of each of the subelements is a 'blank string', that is to say an empty Tag.FormatID.

Procedure for storing a format:

1. A client calls for the storage of the current format by a PUT message on the process variable "SaveFormat“ by passing on a unique FormatID, the format name (optional), the format version and the status.
2. The format management calls for all format-relevant process values by corresponding GET messages and stores the values of these variables under the desired format name. The format is then available.

Procedure for reading out all formats:

1. A client calls for a list of all formulations by a GET message on the process variable "Formats“.
2. The format management sends back an associated data set in the form of a STATE message which contains all of the stored formats.

Note on subformats

The VDMAXML protocol also provides for subformats. These contain only a subset of all format-relevant process variables. A subformat is otherwise treated in exactly the same way as a full format. If a subformat is loaded, any full format already loaded becomes invalid. If a new full format is loaded, any subformat already loaded becomes invalid.

In order to support subformats, the format management must file each format in a component-oriented manner. In this way it is possible, during setup for example, to load only one format for a certain component. The format management then makes available the following process variables for each component:

Memory class (s)	**Name (v)**	**Data**	**Meaning**
cmd	*ComponentID*_SetSubFormat	FormatID FormatName (optional) FormatVersion FormatStatus	Format (formulation) specified by the central format system for the special component (*ComponentID*) For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'
cur	*ComponentID*_CurrentSubFormat	FormatID FormatName FormatVersion FormatStatus	The specified format for the special component (*ComponentID*) If the specified format cannot be correctly loaded, the content of each of the subelements is a 'blank string', that is an empty tag.
cmd	*ComponentID*_SaveSubFormat	FormatID FormatName FormatVersion FormatStatus	Stores the current process values as format. For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'

cur	*ComponentID*_SubFormats	FormatID=0
		FormatName=0
		FormatVersion=0
		FormatStatus
			Sends back all known formats in an associated data set.
			For the element FormatStatus the following states are defined: 'draft', 'released', 'locked' and 'deleted'

The process variable *ComponentID*_SetSubFormat is set by a client with a PUT in order to request a format change on the component defined by ComponentID. The format management sends back, in the form of a STATE message and on the process variable *ComponentID*_CurrentFormat, a confirmation of the success or an indication of the failure of the setting.
In this way subformats can be selectively set on individual components. In doing so, however, the full format for the machine gets always invalid. It must be explicitly set again when a full format is to be loaded.

Examples of data:

```
<VDMAXML_P>
        <PUT v="SetFormat" s="cmd">
                <Format>
                <FormatID>500</FormatID>
                <FormatName>Format 1</FormatName>
                <FormatVersion>1.0</FormatVersion>
                <FormatStatus>released</FormatStatus>
            </Format>
        </PUT>
</VDMAXML_P>


<VDMAXML_P>
        <STATE v="CurrentFormat" s="cur">
            <Format>
                <FormatID>500</FormatID>
                <FormatName>Format 1</FormatName>
                <FormatVersion>1.0</FormatVersion>
                <FormatStatus>released</FormatStatus>
            </Format>
        </STATE>
</VDMAXML_P>


<VDMAXML_P>
        <PUT v="SaveFormat" s="cur">
            <Format>
                <FormatID>500</FormatID>
                <FormatName>Format 1</FormatName>
                <FormatVersion>1.0</FormatVersion>
                <FormatStatus>released</FormatStatus>
            </Format>
        </PUT>
</VDMAXML_P>


<VDMAXML_P>
        <STATE v="Formats" s="cur">
            <Formats>
                <Format>
                    <FormatID>500</FormatID>
                    <FormatName>Format 1</FormatName>
                    <FormatVersion>1.0</FormatVersion>
                    <FormatStatus>released</FormatStatus>
                </Format>
                <Format>
                    <FormatID>501</FormatID>
                    <FormatName>Format 2</FormatName>
                    <FormatVersion>2.0</FormatVersion>
                    <FormatStatus>released</FormatStatus>
                </Format>
            </Formats>
        </STATE>
</VDMAXML_P>


<VDMAXML_P>
        <PUT v="Component1_SetSubFormat" s="cmd">
            <Format>
                <FormatID>500</FormatID>
                <FormatName>SubFormat 1</FormatName>
                <FormatVersion>1.0</FormatVersion>
                <FormatStatus>released</FormatStatus>
            </Format>
        </PUT>
```

```
</VDMAXML_P>


<VDMAXML_P>
        <STATE v="Component1_CurrentSubFormat" s="cur">
                <Format>
                        <FormatID>100</FormatID>
                        <FormatName>SubFormat 1</FormatName>
                        <FormatVersion>1.0</FormatVersion>
                        <FormatStatus>released</FormatStatus>
                </Format>
        </STATE>
</VDMAXML_P>


<VDMAXML_P>
        <PUT v="Component1_SaveSubFormat" s="cur">
                <Format>
                        <FormatID>500</FormatID>
                        <FormatName>SubFormat 2</FormatName>
                        <FormatVersion>1.0</FormatVersion>
                        <FormatStatus>released</FormatStatus>
                </Format>
        </PUT>
</VDMAXML_P>


<VDMAXML_P>
        <STATE v="Component1_SubFormats" s="cur">
                <Formats
                        <Format>
                                <FormatID>500</FormatID>
                                <FormatName>SubFormat 1</FormatName>
                                <FormatVersion>1.0</FormatVersion>
                                <FormatStatus>released</FormatStatus>
                        </Format>
                        <Format>
                                <FormatID>501</FormatID>
                                <FormatName>SubFormat 14</FormatName>
                                <FormatVersion>1.0</FormatVersion>
                                <FormatStatus>released</FormatStatus>
                        </Format>
                </Formats>
        </STATE>
</VDMAXML_P>
```

AuditTrail service

The AuditTrail service is a DataServer. It provides the process variable "AuditTrailMessage“ by means of which the AuditTrail data can be written. This is done as described in the specification. To confirm a logged message, the AuditTrail service sends a STATE message for the process variable "ComponentID_AuditTrailAck“, for the component from which the message originated. The AuditTrail server can receive a series of messages without having to confirm these directly. Through the process variable "AuditTrailCmd“, the AuditTrail server is ordered to start a new log file.
Language coding is by means of subelement "Lang". It should be performed uniformly in accordance with ISO 639-1. Supplements should follow ISO 3166.

Memory class (s)	**Name (v)**	**Data**	**Meaning**
cmd	**AuditTrailCmd**	UserID ComponentID ClientTime	
cur	**AuditTrailMessage**	UserID UserFullName ComponentID MsgID MsgType ClientTime Text Variable (optional)	'MsgID': Unique identification of a message MsgType': Unique identification of the type of message 'ClientTime': Unix time stamp for client's local time when the message is sent 'Text': Content of the message in text form containing the subelements: - TextID - Lang - Entry Here, TextID is the unique identification of the text, 'Lang' is the language code and 'Entry', the text entry. 'Variable': Content of the change of a set variable value containing the subelements: - Name - Storage - Value The subelement 'Value', in turn, contains the subelements 'Old' and 'New' with the old or new value of the variable, respectively.
cur	***ComponentID*_AuditTrailAck**		

Sequence of audit trail message
a) The DataClient sends an audit trail message.
b) The *communicating party* recognizes the message as an audit trail message and provides it with a time stamp in Unix time format in order to ensure that the audit trail message is provided with the synchronized system time.
c) The *communicating party* passes the audit trail message on to the central audit trail server.

d) The AuditTrailServer sets the acknowledge variable and sends a STATE message with the variable ***ComponentID*****_AuditTrailAck.**

Example of data:

```
<VDMAXML_P>
        <PUT v="AuditTrailMessage" s="cur">
                <AuditTrail>
                        <UserID>ADMIN</UserID>
                        <UserFullName>Hans Maier</UserFullName
                        <ComponentID>Panel1</ComponentID>
                        <MsgID>662.1</MsgID>
                        <MsgType>ProductModification</MsgType>
                        <ClientTime>4294967295</ClientTime>
                        <Text>
                                <TextID>ID_PRODUCT_CHANGE</TextID>
                                <Lang>en</Lang>
                                <Entry>User modified product setting</Entry>
                        </Text>
                        <Variable>
                                <Name>SealTemperature</Name>
                                <Storage>cur</Storage>
                                <Value>
                                        <Old>175</Old>
                                        <New>180</New>
                                </Value>
                        </Variable>
                </AuditTrail>
        </PUT>
</VDMAXML_P>
```

EventLog service

The Event Log service is a DataServer which functions in a manner analogous to the AuditTrail service. In the EventLog service, however, there is no confirmation of the received message in the form of a special Ack message.

Memory class (s)	Name (v)	Data	Meaning
cmd	**SysEventCmd**	UserID ComponentID ClientTime	
cur	**SysEventMessage**	UserID UserFullName ComponentID MsgID MsgType ClientTime Text Variable (optional)	'MsgID': Unique identification of a message MsgType': Unique identification of the type of message 'Client time': Unix time stamp for client's local time when the message is sent 'Text': Content of the message in text form containing the subelements: - TextID - Lang - Entry Here, TextID is the unique identification of the text, 'Lang' is the language code and 'Entry', the text entry. 'Variable': Content of the change of a set variable value containing the subelements: - Name - Storage - Value The subelement 'Value', in turn, contains the subelements 'Old' and 'New' with the old or new value of the variable, respectively. The potential contents of subelement "Storage" are specified by the memory classes defined for attribute 's'.

Example of data:

```
<VDMAXML_P>
        <PUT v="SysEventMessage" s="cur">
                <SysEvent>
                        <UserID>ADMIN</UserID>
                        <UserFullName>Hans Maier</UserFullName
                        <ComponentID>Panel1</ComponentID>
                        <MsgID>661</MsgID>
                        <MsgType>UserLogin</MsgType>
                        <ClientTime>4294967321</ClientTime>
                        <Text>
                <TextID>ID_LOGIN</TextID>
                <Lang>en</Lang>
                <Entry>User G. Miller logged in successfully</Entry>
            </Text>
        </SysEvent>
    </PUT>
</VDMAXML_P>
```

BatchControl service

In order to carry out a change of batch consistently for the whole machine it may be necessary to control or interrogate connected devices in a defined sequence (e.g. counter counts).
This purpose is served by three control variables which are to be operated by each connected device producing batch-relevant data. Since the variables are related to devices, their global names start with the respective ComponentID.

Agreed PVs

Memory class (s)	**Name (v)**	**Data**	**Meaning**
cmd	*ComponentID*_BatchCmd		Batch-related command to the component
cur	*ComponentID*_BatchStatus		Message back from the component about the status of the batch-relevant data
cur	*ComponentID*_BatchCmdProgress	0-100	Progress of execution of BatchCmd (in percent)

Commands and status values have yet to be laid down. Typical commands are BatchEnd, BatchReset, BatchRestart.

Appendix IV (informative)
Variable names

The following list is a compilation of the variables defined in this document

Variable	**Meaning**	**Described in Section ...**
*ComponentID*_Start Server	Variable for controlling the DataServer	6.6.2
*ComponentID*_StartScan	Variable for interrogating the current setting values	6.6.2
*ComponentID*_ServerStatus	Variable for transmitting DataServer status messages	6.6.2
*ComponentID*_ServerMessage	Message from a DataServer	6.6.2
LoginRequest	Variable containing data required for registering on the log-on server	Appendix III
LogoutRequest	Variable for logging off on the log-on server	Appendix III
PublicKey	Variable for transmitting the public key in the RSA 1024 bit encryption system	Appendix III
*ComponentID*_UserPerm	Variable containing the log-on result, the userID, the user role and the remaining period of validity of the password	Appendix III
*ComponentID*_UserName	Variable for inquiring about the user currently logged onto a component	Appendix III
SetFormat	Variable containing statement of desired value for format/formulation	Appendix III
CurrentFormat	Variable containing data of the format or formulation currently set	Appendix III
Save Format	Variable for storing the current process values as format	Appendix III
Formats	Variable for reading out all known formats	Appendix III
*ComponentID*_SetSubFormat	Variable containing statement of desired value for format/formulation on the component	Appendix III
*ComponentID*_CurrentSubFormat	Variable containing data of the format or formulation currently set	Appendix III
*ComponentID*_SaveSubFormat	Variable for storing the current process values of a component as format	Appendix III
*ComponentID*_SubFormats	Variable for reading out all known formats of a component	Appendix III
ChangePasswordRequest	Variable containing data needed for a password change	Appendix III
*ComponentID*_ChangePasswordStatus	Variable containing data needed for presenting the result of a password change request	Appendix III
AuditTrailMessage	Variable containing a message to the audit trail server	Appendix III
AuditTrailCmd	Command to the AuditTrail server	Appendix III
*ComponentID*_AuditTrailAck	Acknowledge variable for confirming receipt of an AuditTrail message	Appendix III
SysEventMessage	Variable with content of a message to the event logger	Appendix III
SysEventCmd	Command to the event logger	Appendix III
*ComponentID*_BatchCmd	Batch-related command to the component	Appendix III
*ComponentID*_BatchStatus	Return message from the component about the status of batch-relevant data	Appendix III
*ComponentID*_BatchCmdProgress	Progress of processing ComponentID_BatchCmd in %	Appendix III

Notes:
The name element „*ComponentID_*“ is the uniquely configured name of a component.

Appendix V (informative) Notes on the implementation of a broker architecture

In the VDMAXML_P concept, the broker is not only the central switching point for messages, but it also implements functions without which the overall concept would be difficult to understand.
The broker communicates simultaneously with two types of components: clients (e.g. visualization) and DataServers (e.g. an SPS or a connected device). These two types differ in the nature of their responsibilities and the types of messages allowed. The broker presents itself to the clients as a (TCP) server and to the DataServers as a (TCP) client. In the VDMAXML_P concept, there are some rules which contribute to determining the behavior of the broker:

- All communication relates to process variables (PV).
- A client is responsible for establishing and maintaining its network connections.
- Messages have no explicitly nominated recipient.[5]
- All process variables are configured and known to the broker.[6]
- No dynamic change in the properties of a PV in a DataServer is planned.
- Housekeeping of variables in a server is designed and fixed.[7]

Communication with clients:
The broker accepts connection requests from clients to a ServerPort and processes these requests.
The broker optionally requires authentication from the client. (21 CFR 11)[8]
The broker accepts PUT, GET, SUBS and UNSUBS messages from the clients.
PUT, GET and SUBS messages are passed on to the DataServers in question but not to clients.
An UNSUBS message is sent to a DataServer only when there are no more subscribers for the variable addressed.
If a client connection is interrupted, its subscriptions are canceled internally.
Each PUT, GET and SUBS message for a variable which is either unknown or whose source is not available at the time, is replied to by an "invalidate" message.[9]
The process-global names of the variables are converted to internal item names when they are passed on to the servers.

Communication with servers:
The broker automatically establishes connections to all DataServers known in the system.
From servers, the broker accepts only STATE messages. These are passed on to all subscribers to the variable in question.
In PUT, GET and SUBS messages from clients, the DataServers are addressed in accordance with the RWU attribute of the PV description for the server in question.[10]
When communication with a server fails:

- the broker generates an "invalidate" message for each process variable which the lost server would have to have dealt with (subscriptions only);
- the broker makes regular attempts to reestablish communication with the server; and
- it makes an entry in a log file (limited number).

After the connection is reestablished, all pending subscriptions for the DataServer are renewed. During the outage further subscriptions for the PVs in question are accepted. They are, however, met with an initial "invalidate" message.

[5] Clients receive a message when they have subscribed to the addressed variable; servers receive it when the variable is configured in their variable housekeeping. A sender cannot determine to whom the message is to go.
[6] This applies to memory classes "cur", "fmt" and "mem". Other memory classes are served by fixed servers (log-on) or configured servers.
[7] This applies to memory classes "cur", "fmt" and "mem". Other memory classes are served by fixed servers (log-on) or configured servers.
[8] The sequence has yet to be defined. The feature can be implemented in a protocol-compatible way. Implementation could follow the next version of the protocol.
[9] Agreed error codes could be delivered back here.
[10] Each PV must be configured for each DataServer. The minimum necessary are details about the server-internal variable name (item) and the read/write utility (RWU attribute). If a PV is configured for more than one server, only one server may supply the variable (source, readable). It emerges from the protocol that only PUT messages are sent to more than one server; accordingly, "cross-connections" between two servers cannot arise since servers may not send PUT messages. The configuration files should be standardized so that simple interchange is possible.

The device-internal item names of the variables are converted to the process-global names when they are passed on to the clients.

Routing
The standard routing for the memory classes "cur“, "fmt“ and "mem“ is derived from the configuration data of the DataServers.
PVs of memory class "cmd“ are passed on to the servers only once. They are not stored.[11][12]
PVs of memory class "login“ are passed on only to the configured log-on server.
For all other memory classes a DataServer is explicitly configured. All messages of these classes are then dealt with by only these servers.[13]

Configuration data:
The broker requires the following configuration data:

- its own ServerPort for client inquiries
- the names and IP:Port addresses of all DataServers
- a variable description file for each DataServer specifying name-item relationships and write/read utility[14]
- the names and IP:Port addresses of all DataServers for certain memory classes.

[11] Messages of the "cmd" memory class are commands sent by a client. They are delivered to those servers which have these PVs configured as writable. Commands are transient events. It is not desired for a component coming in later to carry out a past command. Thus, storage of a command is neither necessary nor useful.
[12] The clear differentiation between server and client with regard to the message types supported means that the recipient of a command must be a server. Since server variables are configured, it is also ensured that a command can be passed on only to components which are authorized to execute the command.
[13] The broker concept is highly suitable for the exchange of dynamically altering states, e.g. between two visualization devices. For this purpose, pseudo process variables are set, e.g. in the memory class "visu". Since these have no significance in the real process, their planning requires efforts which are as great as they are superfluous. It is simpler to pass on all messages from a memory class such as "visu" to a single DataServer which can set up the variables dynamically and make them available to subscribers.
[14] Each PV must be configured for each DataServer. The minimum necessary are details about the server-internal variable name (item) and the read/write utility (RWU attribute). If a PV is configured for more than one server, only one server may supply the variable (source, readable). It emerges from the protocol that only PUT messages are sent to more than one server; accordingly, "cross-connections" between two servers cannot arise since servers may not send PUT messages. The configuration files should be standardized so that simple interchange is possible.