VDMA-Fachverbandsschriften
Nahrungsmittelmaschinen und Verpackungsmaschinen

Leitfaden:

Integration von intelligenten Komponenten in Verpackungsmaschinen und Prozessmaschinen im GxP-Umfeld

- Anforderungen aus 21 CFR 11
- Strukturierung des Datenaustauschs unter Berücksichtigung der Anforderungen aus 21 CFR 11
- Standardisierte Kommunikation (VDMAXML_P Version 2.0)
- Beispiele für Implementierung GxP spezifischer Dienste

Nr. 9/2004 Überarbeitete Ausgabe 2012
November 2012

Verband Deutscher Maschinen- und Anlagenbau e.V.

Fachverband Nahrungsmittelmaschinen und Verpackungsmaschinen
Vorsitzender:
Volker Kronseder
Geschäftsführer:
Richard Clemens

Lyoner Straße 18
D-60528 Frankfurt am Main
Telefon+49 69 66 03-14 31
Telefax+49 69 66 03-12 11
E-Mail nuv@vdma.org
Internet www.vdma.org/packtech

VDMA
Technik für Menschen

Integration von intelligenten Komponenten in Verpackungsmaschinen und Prozessmaschinen im GxP-Umfeld

Inhaltsverzeichnis

Diese Publikation wurde vom Arbeitskreis "21 CFR Part 11 - Schnittstellen" der VDMA-Fachabteilung "Maschinen und Anlagen für Pharma und Kosmetik" erarbeitet. Sie ist als Download erhältlich unter WWW.VDMA.ORG/PACKTECH. Anregungen und Ergänzungsvorschläge können an nachstehende Adresse gerichtet werden: Fachverband Nahrungsmittelmaschinen und Verpackungsmaschinen im VDMA, Lyoner Straße 18, 60528 Frankfurt/M., Fax: 069/6603-1211.

Besonderen Dank gebührt Herrn Franz Vater, dessen Vorarbeiten die Grundlage für die Ausführungen in Abschnitt 6 bilden.

Vorbemerkung zur Überarbeitung 2012

2012 wurde diese Fachverbandsschrift einer Überarbeitung unterzogen. Gegenüber der Erstausgabe 2004 wurden folgende Änderungen vorgenommen:

- Redaktionelle Überarbeitung der Einleitung mit dem Ziel, die Kernthematik der Fachverbandsschrift – Einbindung von Komponenten in Maschinen – mehr in den Mittelpunkt zu rücken.
- Auslagerung der spezifizierten Dienste 'Login/Logout'; 'Passwortwechsel', 'Format', 'AuditTrail', 'EventLog' und 'BatchControl' in den Anhang er Fachverbandsschrift. Damit wird verdeutlicht, dass diese spezifizierten Dienste nicht zum verpflichtenden Teil der Spezifikation des Kommunikationsprotoklls VDMAXML_P gehören und somit bei Komponenten, die VDMAXML_P unterstützen die Unterstützung dieser Dienste nicht vorausgesetzt werden kann, sondern ggf. deren Implementierung einer gesonderten Vereinbarung bedarf.
- Der Status des Attributs 's' – Speicherklasse – wird von 'verpflichtend' auf 'optional' geändert. Damit wird der Implementierungspraxis Rechnung getragen.
- Die Versionsnummer von VDMAXML_P wird auf 2.0 gesetzt. Diese Version ist kompatibel mit Version 1.0.

1 Einleitung

Die amerikanische Regulierungsbehörde FDA hat im 21 CFR Part 11 ihre Anforderungen an die Verwendung elektronischer Dokumente und elektronischer Signaturen in der pharmazeutischen Industrie dargelegt. Dies hat Auswirkungen auch auf die Zulieferindustrie. So wird von den Herstellern von Verpackungs- und Prozessmaschinen für die Pharmaindustrie erwartet, dass die Maschinen den Anforderungen aus 21 CFR Part 11 genügen. Bei der Umsetzung dieser Anforderungen kommt der Integration von intelligenten Komponenten eine Schlüsselstellung zu. Hierbei sind grundsätzlich zwei Arten der Einbindung zu unterscheiden: Die Einbindung in eine Maschine und die Einbindung in eine Produktions- bzw. Verpackungslinie. Diese sind schematisch in den Abbildungen 1 und 2 dargelegt.

Abb. 1: Schema Einbindung von intelligenten Komponenten in Maschinen

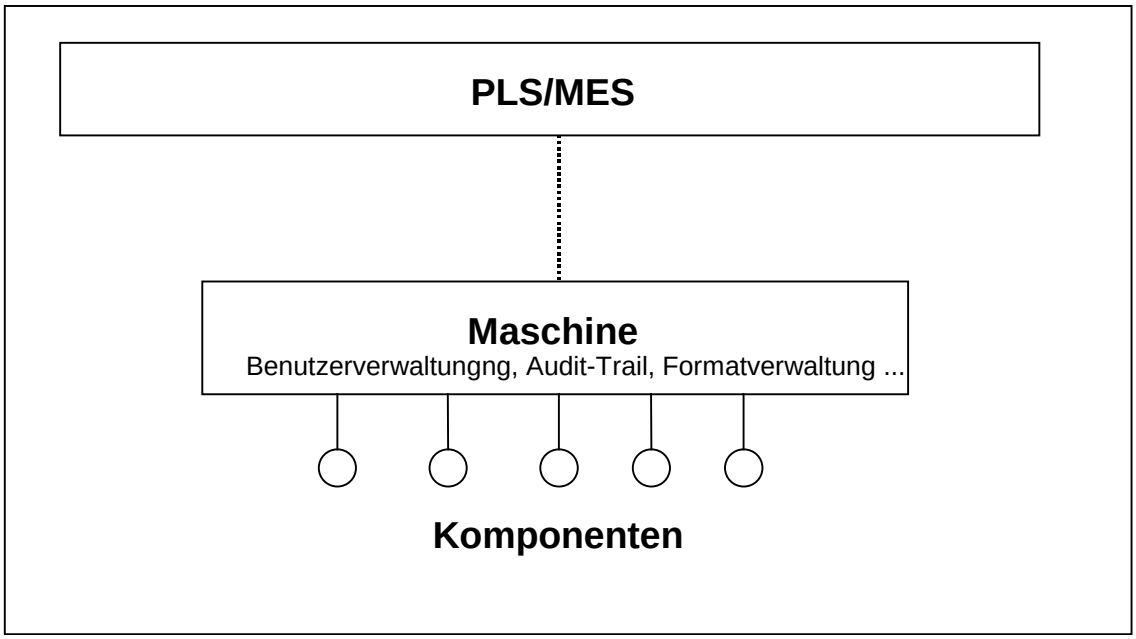
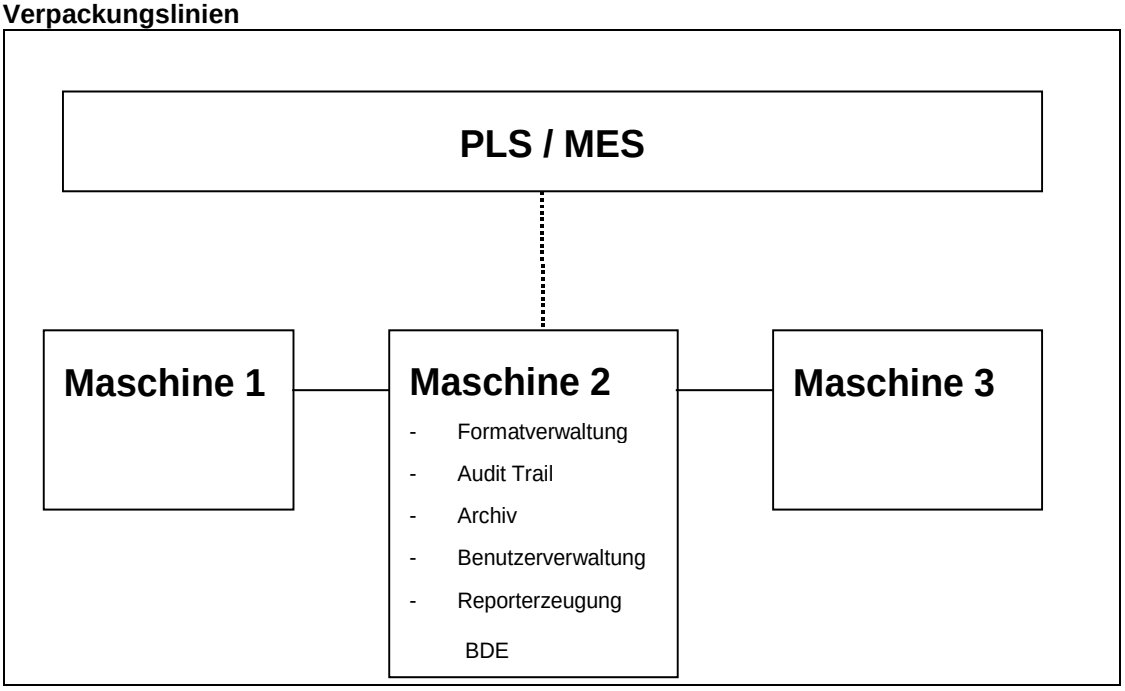


Abb. 2: Schema Einbindung von Maschinen und Komponenten in Produktions- bzw. Verpackungslinien



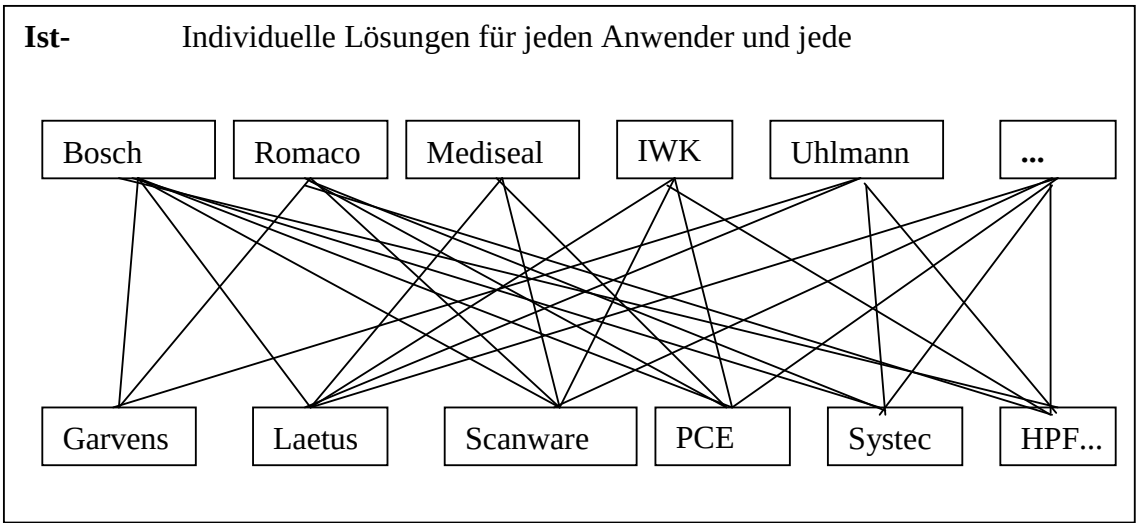
Dieses Arbeitspapier befasst sich mit der Einbindung intelligenter Komponenten in eine Verpackungs- oder Prozessmaschine (Abb.1). Es sei aber darauf hingewiesen, dass sich die in diesem Arbeitspapier dargelegten Überlegungen zum Datenaustausch zwischen Komponenten und Maschine sinngemäß auf die Kommunikation zwischen Maschinen innerhalb einer Linie sowie auf die Kommunikation zwischen Maschine und übergeordnetem PLS/MES übertragen lassen.

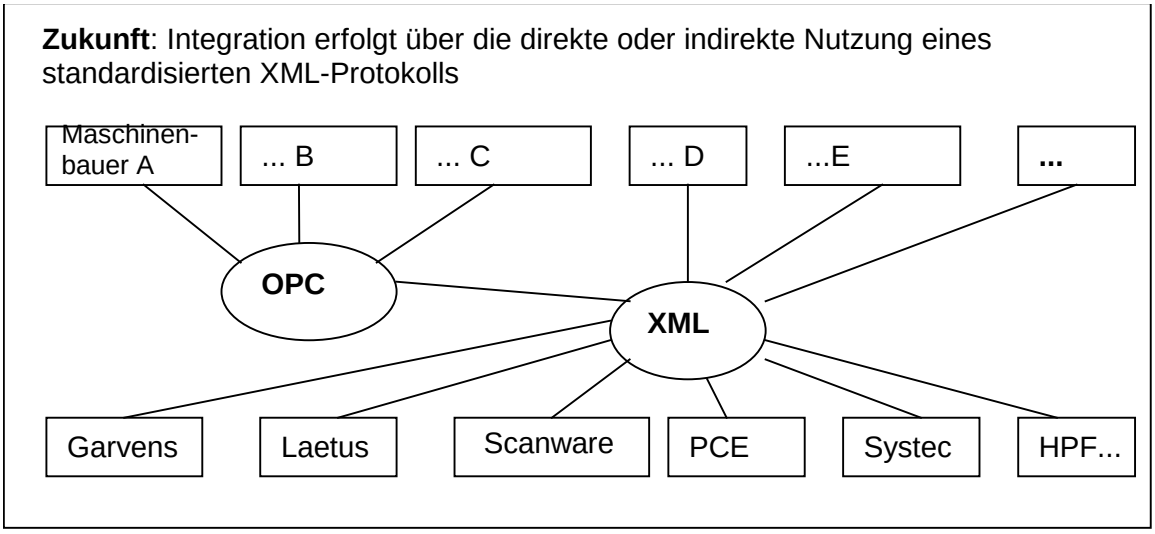
Das hier dargelegte Konzept der Einbindung von Komponenten in eine Maschine verfolgt das Ziel, dass alle das Thema 'electronic records' betreffenden Verantwortlichkeiten bei der übergeordneten Maschine angesiedelt sind (Abschnitt 3). Dieser Ansatz hat den wesentlichen Vorteil, dass der Maschinenhersteller die Übereinstimmung mit den Anforderungen aus 21 CFR 11 unabhängig von der vom Anwender auf Komponentenebene gewählten Konfiguration realisieren kann. Welche Konsequenzen dieses Konzept auf den Datenaustausch zwischen Komponente und Maschine hat, ist in Abschnitt 5 am Beispiel einer einfachen und einer komplexen Komponente dargelegt.

Der Aufwand für die Integration intelligenter Komponenten lässt sich weiter reduzieren, wenn der Datenaustausch zwischen Komponente und Maschine, der zur Gewährleistung der Übereinstimmung mit den Anforderungen aus 21 CFR 11 erforderlich ist, auf der Basis eines standardisierten Protokolls erfolgt. Dies wird durch Abb. 3 illustriert, wobei in dieser Abb. OPC stellvertretend für die Kommunikation über dazwischen liegende Kommunikationsschichten steht. Um dieses Ziel zu erreichen, wird in Abschnitt 6 ein XML-Protokoll mit Namen VDMAXML_P spezifiziert, das die in Abschnitt 3 dargelegten Anforderungen an den Datenaustausch zwischen Komponente und Maschine erfüllt.

Die Kommunikation über VDMAXML_P erleichtert die Umsetzung von Anforderungen in Bezug auf eine zentrale Verwaltung von Benutzerrechten, einem zentralen Audit-Trail oder einer zentralen Formatverwaltung. Diese Fachverbandsschrift definiert zur Unterstützung der Umsetzung dieser Anforderung als optionale Implementierung des Protokolls die Struktur des Datenaustauschs für verschiedene Dienste wie Login/Logout, Passwortwechsel, Formataktivierung, Audit-Trail und Event-Log festgelegt und die dafür benötigten Variablen.(Anhang III (informativ)). Damit wird eine wesentliche Vereinfachung bei der Projektierung dieser Dienste erreicht.

Abb. 3: Komplexitätsreduzierung durch Nutzung standardisierter Protokolle





Das VDMAXML_P-Protokoll ist zur Kommunikation zwischen Komponenten und einer nicht weiter definierten Software, im Folgenden "*Kommunikationspartner*" genannt, konzipiert, die für die Kommunikation mit den weiteren Softwaremodulen der Maschine wie Visualisierung, Formatdatenspeicherung usw. verantwortlich ist. Die Komponente muss dabei nicht wissen, welche Komponenten der Maschine Absender bzw. Empfänger der jeweiligen Nachricht ist. Eine derartige Kommunikationsstruktur bietet Vorteile bei der Implementierung Projekt spezifischer Anforderungen, der nachträglichen Einbindung von Komponenten, der Implementierung zentraler Dienste sowie der Realisierung von Linienleitständen.

VDMAXML_P unterstützt die Realisierung einer Kommunikationsarchitektur auf der Basis von als Nachrichtenbroker fungierenden Softwarekomponenten. Das Vorhandensein einer solchen Brokerarchitektur ist aber nicht Voraussetzung für die Einbindung von Komponenten mittels VDMAXML_P. Anhang V (informativ) enthält Hinweise, die bei der Implementierung einer Brokerarchitektur nach dem VDMAXML_P-Konzept zu beachten sind.

Die im VDMAXML_P-Protokoll spezifizierten Mechanismen können auch zur Kommunikation mit einem übergeordneten Leitsystem verwendet werden. Hierzu ist allerdings eine gesonderte Spezifikation für den Datenaustausch zwischen Maschine und Leitsystemebene erforderlich.

2 Begriffe

Begriff	Erläuterung
Audit-Trail	In Audit-Trail-Dateien werden Ereignisse erfasst, die eine elektronische Aufzeichnung entstehen lassen, verändern oder löschen
DataClient	Eine Komponente (z.B. die Visualisierung), die den aktuellen Inhalt einer PV benötigt. Absender oder Adressat von VDMAXML_P-Nachrichten
DataServer	Eine Komponente (z.B. ein Gerät oder eine Steuerung), die einen Teil zum Variablenhaushalts der Maschine beiträgt. Absender oder Adressat von VDMAXML_P-Nachrichten
Events	Zustandsänderungen von Prozessvariablen oder Definierte Ereignisse, die protokolliert werden sollen.
GxP	Alle Arten von „Good Practices“, z.B „Good Manufacturing Practice“
Hersteller	Der Hersteller der Gesamtmaschine
HMI	Alle Arten von Bedienoberflächen oder Bedienelementen (Human Machine Interface)
intelligente Komponente	Funktionseinheit einer Maschine ausgestattet mit eigener Rechnerleistung und eigener funktionaler Software (einschließlich Funktionen zur Selbstüberwachung und Funktionskontrolle)
Kommunikationspartner	Nicht spezifizierter Empfänger oder Sender von VDMAXML_P-Nachrichten
MES	Management Execution System
OPC	Open Process Control, Kommunikationskonzept im Umfeld von Microsoft Windows.
PLS	Prozessleitsystem
Prozessvariable (PV)	Variable, die der Prozesssteuerung dient und für mehrere Teilsysteme des Gesamtsystems Verpackungs- bzw. Prozessmaschine von Bedeutung ist
TCP-Client	Ein Client im Sinne eines TCP/IP Netzwerks
TCP-Server	Ein Server im Sinne eines TCP/IP Netzwerks
VDMAXML_P-Nachricht	Nachricht im VDMAXML_P-Format
XML	extensible Markup Language
Zulieferer	Der Hersteller einer intelligenten Komponente

3 Anforderungen aus 21 CFR 11

Im Falle der Integration von intelligenten Komponenten in Maschinen liegen die Verantwortlichkeiten für die unter 21 CFR 11 fallenden Funktionen bei der übergeordneten Maschine (Übersicht 1). Damit entfällt in der Regel die Notwendigkeit für die integrierte Komponente die Übereinstimmung mit den Anforderungen aus 21 CFR 11 zu dokumentieren. Aus der Sicht der übergeordneten Maschine sind allerdings die nachfolgenden Anforderungen zu erfüllen, um die Übereinstimmung der Maschine mit 21 CFR 11 feststellen zu können.

- Es ist sicher zu stellen und zu dokumentieren, dass die Sollwerte für die Prozessvariablen der integrierten Komponenten den in Formaten bzw. Rezepturen festgeschriebenen Vorgaben entsprechen.
- Die übergeordnete Maschine muss die Sollwerte von Prozessvariablen für die Steuerung der Komponente an diese übergeben können.
- Die übergeordnete Maschine muss jederzeit verifizieren können, dass die Sollwerte der Prozessvariablen der jeweiligen Komponente den Vorgaben der aktuellen Format- bzw. Rezepturvorgabe entsprechen. (Format- bzw. Rezepturabgleich)
- Können die Sollwerte der Prozessvariablen direkt an der Komponente über ein eigenes HMI geändert werden, ist sicher zu stellen, dass diese Änderungen nur durch autorisierte Nutzer vorgenommen werden. Die Nutzerverwaltung erfolgt dabei zentral über die übergeordnete Maschine (auch bei Login auf lokalem HMI). Diese Änderungen müssen an die übergeordnete Maschine gemeldet werden.

- Bei der Komponente erfolgt keine dauerhafte Speicherung qualitätsrelevanter Daten (z.B. Batchprotokolle)
- Die Komponente führt keinen eigenen Audit-Trail

Übersicht 1: Schematische Darstellung der Verantwortlichkeiten

	Komponente	Maschine[3)]
Nutzerverwaltung	-	x
Login	(x)[1)]	x
Rezeptur- und Formatverwaltung	-	x
Audit-Trail	-	x
Meldung von Änderungen von qualitätsrelevanten Sollwertvorgaben	(x)[2)]	x
dauerhafte Dokumentation von Prozessdaten (Archiv)	-	x
Reportgenerator		x

1) Sofern der Login dazu berechtigen soll, qualitätsrelevante Sollwerte abzuändern, hat dieser unter Nutzung der zentralen Nutzerverwaltung der Maschine zu erfolgen.
2) Sofern über ein lokales HMI Sollwertvorgaben geändert werden können, ist sicher zu stellen, dass diese nur von autorisierten Personen vorgenommen werden kann und die Änderung an die übergeordnete Maschine gemeldet wird.
3) Die in diesem Arbeitspapier dargelegten Ausführungen zum Datenaustausch zwischen Komponente und Maschine gelten auch dann, wenn die aufgeführten Verantwortlichkeiten von der Maschine auf die PLS/MES-Ebene übertragen werden.

4 Allgemeine technische Anforderungen für den Datenaustausch zwischen Komponente und Maschine

- Möglichst geringe Hardwarevoraussetzungen
- Einfache Schnittstelle
- Einfaches Protokoll
- Erweiterbares Protokoll
- Niedrige Netzlast
- Datenübertragung im Klartext
- Sichere Übertragung und Überwachbarkeit der Kommunikation
- Schnelle Übertragung kleiner Datenmengen
- Geringe Kopplung
- Geringe Komplexität
- Offen gelegte, herstellerneutrale und vom Betriebsystem unabhängige Lösung
- Aufsetzen auf bekannte Industriestandards
- Skalierbarkeit auf unterschiedliche Komplexitätsgrade von Komponenten
- Geringer Integrationsaufwand

5 Datenaustausch zwischen Maschine und intelligenten Komponenten

5.1 Komplexitätsgrad einer Komponente

Merkmale der Komponente	trifft zu für Komponenten niedriger Komplexität	trifft zu für Komponenten hoher Komplexität
Login an der Komponente	nein	ja
hohe Anzahl komponenten-eigener Prozessvariablen	nein	ja
lernendes System mit Formatmanagement	nein	ja
Auslagerung von Daten in Archiv	nein	ja
Meldung GxP-relevanter Ereignisse	nein	ja
eigener Audit-Trail	nein	möglich, bei Einbindung in Maschinen nicht zugelassen
eigener Reportgenerator	nein	möglich, bei Einbindung in Maschinen nicht zugelassen

5.2 Datenaustausch zwischen Maschine und einer komplexen Komponente am Beispiel eines Kamerasystems

Ein Kamerasystem ist dadurch gekennzeichnet, dass es sich um einen lernenden Sensor handelt, dessen Referenzparameter aus nicht vordefinierbaren Werten bestehen. Zur Wahrnehmung der ihm zugedachten Funktion ist das System in der Regel mit einer eigenen Bedienoberfläche ausgestattet. Ein weiteres Kennzeichen ist, dass viele der Prozessvariablen des Systems formatabhängig sind. Welche Konsequenzen ergeben sich aus diesen Merkmalen komplexer Systeme für den Datenaustausch zwischen Komponente und Maschine unter GxP-Gesichtspunkten?

Login:
Der Login für die Nutzung der Bedienoberfläche der Komponente erfolgt an der Komponente selbst. Dies gilt auch für die Identifikation des Nutzers. Nach erfolgter Identifikation erfolgt die Anmeldung des Nutzers bei der zentralen Nutzerverwaltung der übergeordneten zentralen Nutzerverwaltung mit Rückmeldung der für den Nutzer eingetragenen Benutzerrechte.

Laden von formatabhängigen Sollwerten von Prozessvariablen:
Wegen der Vielzahl der formatabhängigen Sollwerte von Prozessvariablen werden diese geräteintern verwaltet. Diese für die einzelnen Formate geltenden Grundeinstellungen werden im Rahmen der Qualifizierung dokumentiert. Für GxP-Zwecke ist es ausreichend, dass die übergeordnete Maschine den Namen und die aktuelle Version des zu ladenden Formats übergibt und die Komponente die Übereinstimmung des vorgegebenen Formats mit dem geladenen Format zurückmeldet. Werden während des Betriebs Änderungen der Grundeinstellungen vorgenommen, sind diese an den *Kommunikationspartner* zu kommunizieren.

Änderungen von Basiseinstellwerten (formatneutrale Grundeinstellungen) im Rahmen des Konfigurationsmanagements:
Die Änderungen der Basiseinstellwerte werden im Rahmen des Change-Control-Prozesses des Betriebes dokumentiert und sind daher nicht Gegenstand des Audittrails der Maschine.
Beispiel: Wird ein DMS-basierter Kraftaufnehmer durch einen piezoelektrischen ersetzt, bleiben die Sollwerte des Formats davon unbeeinflusst. Gleichzeitig können allerdings Einstellungsparameter der Software betroffen sein. Dies wird zusammen mit den Umbaumaßnahmen an der Anlage im Rahmen des ChangeControls dokumentiert und validiert.

Archivierung:
In der Komponente erfolgt keine dauerhafte Speicherung GxP-relevanter Dateien oder Daten. Soweit Daten oder Dateien, die von der Komponente erzeugt werden, archiviert werden sollen, sind diese in geeigneter Form dem zentralen Archivserver zu übergeben. Die Übergabe wird über den Audit-Trail dokumentiert.

Audit-Trail:
Die Verantwortung für den Audittrail liegt bei der übergeordneten Maschine. Die Komponente meldet GxP-relevante Ereignisse. Es wird kein Audittrail seitens der Komponente geführt.

Reportgenerator:
Die Komponente erstellt keine GxP-relevanten Reports. Daten und Dateien, die die übergeordnete Maschine zur Erzeugung von Reports benötigt, werden von der Komponente auf Anforderung bereit gestellt.

5.3 Datenaustausch zwischen Maschine und Komponenten von geringer Komplexität am Beispiel eines Temperaturfühlers

Im Vergleich zu einem Kamerasystem ist ein Temperaturfühler ein einfaches System. Es misst die Temperatur und meldet diesen Istwert an die übergeordnete Maschine. Eine GxP-gerechte Produktion setzt die Kalibrierung des Temperaturfühlers voraus.

Login:
Ein Login an der Komponente ist nicht vorgesehen

Laden von formatabhängigen Sollwerten von Prozessvariablen:
GxP-relevante Sollwerte und Toleranzen von Prozessvariablen werden nicht geräteintern verwaltet, sondern direkt von der übergeordneten Maschine an die Komponente übergeben. Die übergeordnete Maschine muss jeder Zeit die eingestellten Sollwerte der GxP-relevanten Prozessvariablen der Komponente verifizieren können.

Archivierung:
Eine Archivierungsfunktion ist nicht vorgesehen.

Audit-Trail:
Es wird kein Audit-Trail bei der Komponente geführt. Die Komponente meldet den Ist-Wert des Messwertes an die übergeordnete Maschine.

Reportgenerator:
Die Erzeugung GxP-relevanter Reports durch die Komponente ist nicht möglich.

6 Standardisierte Kommunikation zwischen Maschine und Komponenten

Die Kommunikation zwischen Maschine und ihren Komponenten hat verschiedene Ebenen, die in Anlehnung an das ISO/OSI-Schichtenmodell beschrieben werden sollen.

Schicht	Name	Realisiert durch	Bedeutung
7c	Anwendung	Programme und Konventionen	Dienste wie Login, BatchControl, FormatManagement usw.
7b		Besondere Prozessvariablen	Definition besonderer Prozessvariablen und Nachrichten-Inhalte
7a		Prozessvariablen	Austausch von Prozessvariablen
6	Darstellung	VDMAXML_P-Nachricht	Konventionen über den Aufbau der Nachricht
5b	Kommunikationssteuerung	Broker, DataClients, DataServer	Vereinbarungen darüber wie der Broker , DataClients und Datenquellen Nachrichten verarbeiten
5a		Sockets	Nachrichtenübertragung
4	Transport	TCP	
3	Vermittlung	IP	
2	Sicherung	Ethernet	
1	Bitübertragung		

Die Zuordnung der Schichten 5 und 6 dient im Wesentlichen der Strukturierung. Die Schicht 5 umfasst die eigentliche Nachrichtenübertragung über TCP-Sockets (5a) und das vereinbarte Verhalten der beteiligten *Kommunikationspartner* (5b).

6.1 Kommunikationsschichten 1 bis 4

Für die Schichten 1-4 wird ein TCP/IP-Netzwerk auf einer Ethernet-Physik festgelegt. Komponenten, die nicht über eine TCP/IP fähige Ethernetschnittstelle verfügen und/oder das VDMAXML_P-Protokoll nicht implementiert haben, müssen indirekt über DataServer eingebunden werden.

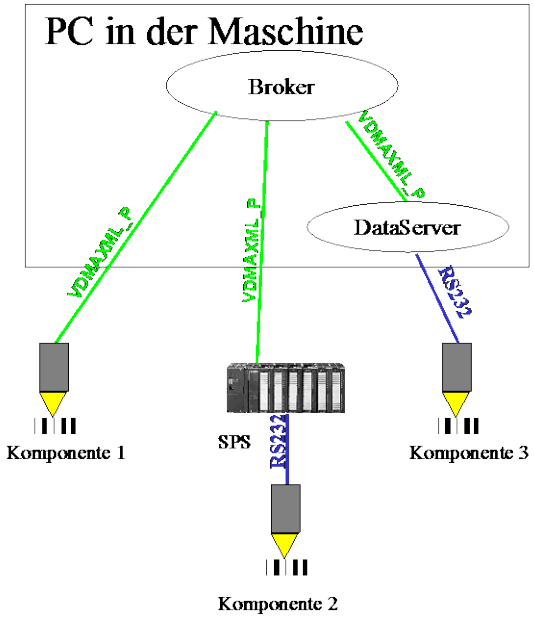


Abb. 5 Indirekte Einbindung von nicht VDMAXML_P-fähigen Geräten

Abb. 5 stellt zwei Möglichkeiten der Einbindung eines Gerätes dar, das nur über eine RS232-Schnittstelle verfügt: Die Komponente 2 ist an die RS232-Schnittstelle (oder I/Os) einer SPS angeschlossen. Die Daten der Komponente 2 liegen als Kopie in SPS-Variablen vor, auf die über VDMAXML_P zugegriffen wird. Die SPS sendet bzw. empfängt die Daten der Komponente über die RS232 Schnittstelle und hat damit eine Stellvertreterrolle. In der zweiten Variante ist die RS232-Schnittstelle der Komponente 3 an den PC angeschlossen, auf dem eine DataServer genannte Software läuft, die zum *Kommunikationspartner*,- z.B. ein Nachrichtenbroker - hin das VDMAXML_P Protokoll implementiert und die Kommandos auf die RS232-Schnittstelle umsetzt. Es kann eine reine Umsetzung auf eine andere Übertragungsphysik (RS232) sein oder auch die inhaltliche Umsetzung auf ein proprietäres Protokoll der Komponente 3.

6.2 Kommunikationsschicht 5a

Die Übertragung erfolgt über einfache TCP-Sockets. DataServer und Geräte stellen TCP-Server-Sockets zur Verfügung, die der Broker ansprechen kann.
Der Broker stellt TCP-Server-Sockets zur Verfügung, die DataClients wie die Visualisierung oder andere Programme ansprechen können.
Die Identifikation einer Komponente ist ihre IP-Adresse und ihr TCP-Server-Port.

6.3 Kommunikationsschicht 5b

Diese Schicht legt fest, wie sich die Kommunikationsteilnehmer verhalten sollen:

- Jeder TCP-Client ist für seine Verbindungen verantwortlich. Bricht die Verbindung ab, so muss der TCP-Client regelmäßig versuchen, sie wieder aufzubauen.
- Jeder TCP-Server sorgt dafür, dass er ansprechbar ist.
- Jeder TCP-Server ist für untergeordnete Geräte und Verbindungen verantwortlich.

- Im Falle der Implementierung einer Brokerarchitektur kennt der Broker alle Prozessvariablen und übernimmt das Routing der Nachricht. Er sorgt dafür, dass alle Abonnenten und alle anderen registrierten Datensenken von Zustandsänderungen einer Variablen informiert werden.(s. auch Anhang V)

6.4 Kommunikationsschicht 6: Standardisiertes XML-Protokoll für die Kommunikation zwischen Maschine und Komponente

Nachfolgend wird ein nachrichtenbasiertes Kommunikationskonzept für den Datenaustausch zwischen Komponente und Maschine dargelegt. Adressat aller Nachrichten ist eine Softwarekomponente der Maschine, im Folgenden vereinfachend *Kommunikationspartner* genannt.. Diese Softwarekomponente kann ein Nachrichtenbroker sein, der die endgültigen Empfänger der Nachrichten festlegt und die Nachrichten an diese weiter leitet. Die Funktion eines solchen Brokers im Rahmen dieses Kommunikationskonzeptes ist im Anhang V näher erläutert.

Der Absender der Nachricht muss nicht wissen, welche funktionalen Einheiten der Maschine die ausgesandte Nachricht letztlich verarbeiten. Die Kommunikation der Komponenten mit der Maschine erfolgt über das standardisierte XML-Protokoll 'VDMAXML_P'.

6.4.1 Aufbau von VDMAXML_P-Nachrichten

Alle VDMAXML_P-Nachrichten sind Dokumente entsprechend den XML-Konventionen. Darüber hinaus gelten folgende Vereinbarungen:

1. Jedes Dokument besteht aus einem Wurzelelement <VDMAXML_P> und einem darin enthaltenen Element mit dem eigentlichen Nachrichteninhalt.
2. Das einzige Unterelement des Elements VDMAXML_P hat den Namen der Funktion der Nachricht (z.B. STATE, PUT).
3. Die notwendigen Informationen (außer den Daten) werden als Attribute transportiert.
4. Die Daten sollen kurz sein (keine großen Felder oder Texte). Die Gesamtlänge des Strings sollte üblicherweise 2000 Bytes nicht übersteigen[1]
5. Dokumente müssen im UTF-8 Encoding übertragen werden
6. Der Inhalt eines Elementes besteht entweder aus einer Zeichenkette oder ein oder mehreren Elementen. Im zweiten Fall sind zur Strukturierung Leerzeichen, Tabulatorzeichen oder Zeilenvorschübe (Cr und/oder LF) zugelassen.

[1] Die maximale Länge sollte durch den Hersteller der Maschine in Abhängigkeit des verwendeten Brokers spezifiziert werden.

6.4.2 Arten von VDMAXML_P-Nachrichten

Das Unterelement des Wurzelelements enthält die eigentliche Nachricht. In den folgenden Ausführungen wird der Begriff „Nachricht“ in der Regel als Synonym für dieses Element verwendet. Folgende Arten von Nachrichten werden unterschieden:

Nachrichtentyp	**Bedeutung**
STATE	Meldet den aktuell neuen Zustand einer Variablen
GET	Fordert den *Kommunikationspartner* oder DataServer auf, den Inhalt einer Variablen zu melden
PUT	Fordert den *Kommunikationspartner* oder DataServer auf, eine Variable auf einen bestimmten Wert zu setzen
SUBS	Abonniert beim *Kommunikationspartner* oder einem DataServer eine Variable. Immer wenn sich die Variable ändert soll der DataServer den aktuellen Wert der PV melden. Der DataServer liefert auf jeden Fall den Wert der Variablen erstmals beim Aufruf des Subscribe-Befehls.
UNSUBS	Kündigt das Abonnement auf diese Variable

Anmerkung:
Alle Nachrichtentypen müssen auf einer Komomponente implementiert sein.

6.4.3 Attribute einer VDMAXML_P-Nachricht

In VDMAXML_P-Nachrichten dürfen nur die nachfolgend spezifizierten Attribute verwendet werden. Nur wenige davon müssen von allen Komponenten verstanden werden. Wird eines dieser Attribute von einer Komponente nicht verstanden, soll es ignoriert werden.

Attribut	**Name**	**Bedeutung**	**Zwingend**	**Anmerk.**
v	variable	Name der Variablen	X	1
s	storage	Speicherklasse, zur Zeit: cur (current, aktueller Wert) mem (temporär gespeicherter Wert) fmt (Formatspeicher) cmd (Kommando) state (Statusmeldung) visu (reserviert für Visualisierung) vds (DataServer Steuerung) login (reserviert für Login-Anfragen) msg(message, wird nicht vom Kommunikationspartner gespeichert)		8
f	from	*ComponentID* des Absenders		7
t	time	(UNIX) Zeitstempel		2
k	kind	Art der Daten, phys. Grösse, z.B.: Temperatur, Druck		3
u	unit	verwendete physikalische Einheit, z.B. mm, inch, Torr		4
c	command	definierte eingebettete Kommandos, z.Zt.: i = invalidate		6
d	data type	Datentyp, z.B.: i1, ui4, number		5

Anmerkungen:
1) Namen für Prozessvariable müssen vom Hersteller vergeben bzw. mit diesem vereinbart werden. Sie gelten also global.

2) Der Zeitstempel wird als dezimale Zahlendarstellung einer 32-bit-Variablen nach UNIX- Konvention erwartet und gesendet. Der Zeitstempel ist optional. DataServer sollten keinen Zeitstempel senden, wenn die Zeiten der Komponenten nicht exakt synchronisiert sind (durch eine Netzwerk-Zeit). Wenn die exakte Zeit relevant ist, gilt im Zweifel die Zeit des *Kommunikationspartner*s der Komponente als Referenz.

3) Die Art der Grösse dient der Konsistenzprüfung und kann optional angegeben werden. In der Regel wird die Angabe aber zur Laufzeit nicht verwendet. Es sollen die international vereinbarten Grössen des SI-Systems verwendet werden.

4) Die verwendete Maßeinheit sollte angegeben werden, wenn nicht sicher ist, dass die Maßeinheit der System-Maßeinheit für diese Prozessvariable enspricht. Wenn möglich sollten die international vereinbarten Grössen des SI-Systems verwendet werden.

5) Das Datenformat bezieht sich auf die Realisierung im Gerät und ist unabhängig von der globalen Definition der Prozessvariablen. Es kann zur Konsistenzprüfung verwendet werden. Es sollten definierte Datentypen verwendet werden.

6) Diese Kommandos bieten die Möglichkeit, Steuerinformationen in den Datenstrom einzubetten. Bisher sind definiert: c="i" (invalidate). Damit soll ein DataServer **mitteilen**, dass die genannte PV ungültig ist, z.B. weil ihr Zustand noch nicht festgestellt werden konnte oder die Kommunikation zum Gerät gestört ist.
Ein Kommunikationspartner muss auch ein leeres Attribut "c" ohne Störungen der Kommunikation verarbeiten können.

7) Die optionale ComponentID soll gesetzt sein, wenn die Identifizierung des Absenders für einen Dienst relevant ist (z.B. LoginRequest).

8) Die Storage Klasse 's' ist insbesondere für eine Broker Architektur vorgesehen und beeinflusst das Nachrichtenrouting in der Maschine.

6.4.4 Beispiel für eine VDMAXML_P-Nachricht

Im Folgenden ist der Aufbau für die Nachricht gezeigt, mit der eine Komponente anzeigt, dass sich die Prozessvariable "FillLevel" auf den Wert "305 (mm)" geändert hat. Dies geschieht mittels einer "STATE" (Status-) Meldung. Nach dem gleichen Muster sind alle Nachrichten aufgebaut. Lediglich das Tag "STATE" wird jeweils ersetzt durch die Bedeutung der Nachricht, z.B. PUT oder GET.

Bsp.:

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

Bedeutung:

<VDMAXML_P>	*das umschliessende Element einer VDMAXML_P-Nachricht*
<STATE	*das eröffnende Tag einer STATE-Meldung*
t="99458543"	*Zeitstempel*
s="cur"	*Speicherklasse "cur" = current = aktueller Messwert*
v="FillLevel"	*Name der Prozessvariablen*
u="mm"	*Messwert ist in Millimetern angegeben*
>305<	*Füllhöhe zum Zeitpunkt der Meldung = 305 mm*
/STATE>	*das schliessende Tag der Meldung*
</VDMAXML_P>	*das schliessende Tag einer VDMAXML_P-Nachricht*

Die Nachricht darf Leerzeichen, Tabulatorzeichen und Zeilenvorschübe (CR und/oder LF) enthalten.

6.4.5 Rolle von DataClients und DataServern im System

Ob eine Komponente die Rolle eines DataClient oder eines DataServers im System darstellt, hängt von der jeweiligen Funktionalität ab, die von der Komponente bereitgestellt oder genutzt wird.

Stellt die Komponente Prozessvariablen zur Verfügung oder stellt sie Dienste der Schicht 7c bereit ist sie ein DataServer.

Ist die Komponente an Variablen einer anderen Komponente interessiert oder nutzt sie einen der Dienste der Schicht 7c so ist sie ein DataClient.

Stellt eine Komponente eigene Prozessvariablen zur Verfügung und benötigt selbst Prozessvariablen von anderen Komponenten oder möchte Dienste der Schicht 7c ansprechen, so muss sie sowohl DataServer als auch DataClient gleichzeitig sein. D.h. die Komponente muss

sowohl die im Folgenden spezifizierten DataClient als auch DataServer Funktionalitäten implementieren.

Durch diese verbindliche Definition ist festgelegt, welche Funktionalität eine Komponente implementieren muss, um in dem Verbund erfolgreich mit den anderen Komponenten kommunizieren zu können.

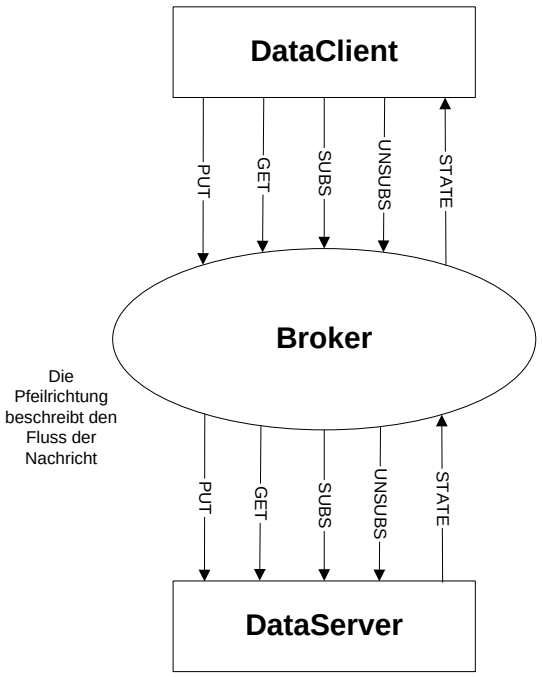


Abb. 6: Kommunikation zwischen DataClient und DataServer über einen Broker

6.4.5.1 DataClient

Von einem DataClient dürfen nur die folgenden in Kapitel 6.4.2 festgelegten Nachrichten gesendet werden:

- GET
- PUT
- SUBS
- UNSUBS

Ein DataClient darf nur STATE Nachrichten verarbeiten. Alle anderen Nachrichten sind nicht zulässig. Ein DataClient darf auf eine ungültige Nachricht nicht reagieren, er muss jedoch sicherstellen, dass er weiterhin korrekt funktioniert. Er sollte die ungültige Nachricht protokollieren. Er verarbeitet nur STATE Nachrichten, die er entweder als Antwort eines gesendeten GET erhält, oder die er in nicht definierten Intervallen vom *Kommunikationspartner* durch ein vorher ausgeführtes Abonnement (SUBS) erhält.

Ein DataClient darf keine STATE Nachricht schicken, da diese immer nur den Zustand einer Prozessvariablen anzeigt und ein Client keine eigenen Prozessvariablen besitzt.

Ein DataClient ist der Initiator einer Kommunikation. Er schickt GET, PUT, SUBS oder UNSUBS Nachrichten an den *Kommunikationspartner*, der diese dann an den entsprechenden DataServer oder Dienstanbieter weiterleitet. Dieser wiederum schickt eine Antwort in Form einer STATE Nachricht für ein GET und ein SUBS. Im Falle einer SUBS Nachricht erhält der DataClient dann für jede Änderung der abonnierten Variablen eine entsprechende STATE Nachricht, die vom DataServer initiiert wurde. Erhält er keine Antwort innerhalb einer definierten Zeit (Timeout) kann er davon ausgehen, dass ein Fehler im System vorliegt.

Auf eine PUT Nachricht erhält der Client keine direkte Antwort zurück. Der DataServer, der die Nachricht empfängt, wird versuchen, die Prozessvariable zu setzen und daraufhin eine STATE Meldung an das System senden, entweder mit dem neu gesetzten Prozesswert oder mit „invalidate“, falls der Wert aus irgendwelchen Gründen nicht gesetzt werden konnte. Damit der Client, der das PUT initiiert hat, diese Änderung mitbekommt, bzw. das PUT überprüfen kann, hat er prinzipiell 2 Möglichkeiten:

- Er sendet ein explizites GET für diese Variable um den Wert nochmals abzufragen
- Er hat sich vor dem PUT durch ein SUBS für Werteänderungen dieser Prozessvariablen registriert.

Für eine UNSUBS Nachricht erhält der DataClient keine Antwort. Er erhält im Folgenden keine weiteren STATE Meldungen für diese Variable. Durch das zeitlich nicht definierte Verhalten kann es aber durchaus möglich sein, das der DataClient nach dem Absenden einer UNSUBS Meldung noch STATE Meldungen erhält, wenn diese beim *Kommunikationspartner* bereits vor dem Absenden des UNSUBS aufgelaufen sind, aber erst danach verteilt werden. Der DataClient muss hierauf vorbereitet sein und darf nicht mit einem Fehler reagieren.

Einen Fehlerfall kann der Client dem Benutzer anzeigen, er muss jedoch sicherstellen, weiterhin korrekt zu funktionieren.

Soll eine Komponente zusätzlich DataServer Funktionalität bereitstellen, muss sie auch DataServer Funktionalität implementieren (siehe 6.4.5.2)

6.4.5.2 DataServer

Von einem DataServer dürfen nur STATE Nachrichten verschickt werden. Ein DataServer schickt niemals eine GET, PUT, SUBS oder UNSUBS Nachricht. Ein DataServer verschickt STATE Nachrichten nur für Prozessvariable, die bei Ihm als abonniert gekennzeichnet sind oder mittels GET angefordert werden. Ein Abonnement liegt dann vor, wenn der DataServer für eine Prozessvariable eine SUBS Nachricht vom *Kommunikationspartner* erhält.

Ein DataServer darf hingegen nur GET, PUT, SUBS und UNSUBS Nachrichten empfangen. Er darf auf ungültige Nachrichten nicht reagieren und muss sicherstellen, weiterhin korrekt zu funktionieren.

Ein DataServer kennt keine Referenzzählung bei SUBS und UNSUBS Nachrichten. Erhält er eine SUBS Nachricht für eine Prozessvariable, markiert er diese als abonniert und schickt daraufhin für jede Änderung der Prozessvariable eine STATE Nachricht an den *Kommunikationspartner*. Weitere SUBS Nachrichten werden vom DataServer mit einer STATE Nachricht beantworten, die den aktuellen Wert der Variablen enthält. Bei einer UNSUBS Nachricht wird die Variable wieder als nicht abonniert gekennzeichnet, weitere UNSUBS Nachrichten werden vom DataServer ignoriert.

Bei einem Verbindungsabbruch oder Kommunikationsfehler zum *Kommunikationspartner* löscht der DataServer alle Abonnements und stellt sicher, dass er aus seinem Server-Port wieder erreichbar ist. Anschließend wartet er darauf, dass der *Kommunikationspartner* eine Verbindung zu ihm aufbaut.

Soll eine Komponente zusätzlich DataClient Funktionalität bereitstellen bzw. benötigen, muss sie auch DataClient Funktionalität implementieren (siehe 6.4.5.1)

6.4.6 Definition des Nachrichtenflusses

Im Folgenden wird der Nachrichtenfluss jeder Nachricht beschrieben.

Allgemein gilt, dass für jedes nicht ausführbare Kommando ein Invalidate in Form einer STATE Meldung geschickt wird und zwar von der Komponente, die den Fehler als erstes feststellen kann. Dies kann der *Kommunikationspartner* sein, oder der DataServer, für den das Kommando bestimmt ist.

Eine globale Prozessvariable existiert für die Version 2.0 der Spezifikation immer genau in einem DataServer. Es gibt keine zwei DataServer, von denen die gleiche globale Variable angeboten wird.

6.4.6.1 Nachrichtenfluss bei PUT-Nachrichten

Eine PUT Nachricht wird immer vom DataClient initiiert. Der Sinn der Nachricht ist es, eine Prozessvariable auf einen definierten Wert zu setzen.

1. Der Client schickt die PUT Nachricht an den *Kommunikationspartner*
2. Der *Kommunikationspartner* identifiziert den DataServer, der die Prozessvariable zur Verfügung stellt und leitet die PUT Nachricht an diesen weiter. Durch die Projektierung ist sicher zu stellen, dass nur ein DataServer als Quelle für eine Prozessvariable benannt ist.
3. Der DataServer empfängt die Nachricht und setzt die geforderte Prozessvariable auf den in der Nachricht angegebenen Wert.
4. Falls für diese Variable ein Abonnement existiert, sendet er eine STATE Nachricht an den *Kommunikationspartner*. Die STATE Nachricht enthält den aktuellen, geänderten Wert der Prozessvariablen oder „invalidate“, falls der Wert nicht gesetzt werden konnte.
   Falls kein Abonnement existiert, ist der Nachrichtenfluss hiermit beendet.
5. Der *Kommunikationspartner* leitet die STATE Nachricht an alle DataClients weiter, die sich für diese Prozessvariablen mit einer SUBS Nachricht registriert haben.

6.4.6.2 Nachrichtenfluss bei Get-Nachrichten

Eine GET Nachricht wird immer vom DataClient initiiert. Der Sinn der Nachricht ist es, den aktuellen Wert einer Prozessvariablen explizit abzufragen.

1. Der Client schickt die GET Nachricht an den *Kommunikationspartner*
2. Der *Kommunikationspartner* identifiziert den DataServer, der die Prozessvariable zur Verfügung stellt und leitet die GET Nachricht an diesen weiter.
3. Der DataServer empfängt die Nachricht, liest den aktuellen Wert dieser Variablen und sendet eine STATE Nachricht an den *Kommunikationspartner*. Die STATE Nachricht enthält den aktuellen Wert der Prozessvariablen oder „invalidate“, falls der Wert der Prozessvariablen momentan ungültig oder nicht gesetzt ist.
4. Der *Kommunikationspartner* leitet die Nachricht an den DataClient weiter, der die GET Nachricht initiiert hat und an alle DataClients, die sich für diese Prozessvariablen mit einer SUBS Nachricht registriert haben, auch wenn sich der Wert der Prozessvariablen nicht geändert hat.

6.4.6.3 Nachrichtenfluss bei SUBS-Nachrichten

Eine SUBS Nachricht wird immer vom DataClient initiiert. Der Sinn der Nachricht ist es, sich für eine Prozessvariablenänderung zu registrieren.

1. Der Client schickt die SUBS Nachricht an den *Kommunikationspartner*
2. Der *Kommunikationspartner* identifiziert den DataServer, der die Prozessvariable zur Verfügung stellt und leitet die SUBS Nachricht an diesen weiter. Zusätzlich registriert er intern den DataClient als Abonnent dieser Variablen. Alle zukünftigen STATE Nachrichten, die der *Kommunikationspartner* für diese Variable empfängt wird er an diesen DataClient weiterleiten bis er eine entsprechende UNSUBS Nachricht des Clients enthält.
3. Der DataServer empfängt die Nachricht und registriert intern diese Variable als abonniert. Für jede zukünftige Änderung dieser Variablen initiiert der DataServer eine STATE Nachricht und sendet diese an den *Kommunikationspartner*.
4. Der DataServer schickt als Antwort auf die SUBS Nachricht eine initiale STATE Nachricht an den *Kommunikationspartner*. Die STATE Nachricht enthält den aktuellen Wert der Prozessvariablen oder „invalidate“, falls der Wert momentan ungültig oder nicht gesetzt ist.
5. Der *Kommunikationspartner* leitet die STATE Nachricht an alle DataClients weiter, die sich für diese Prozessvariable mit einer SUBS Nachricht registriert haben.
6. Weitere SUBS Nachrichten für diese Variable werden vom Server mit einer STATE Nachricht beantwortet

6.4.6.4 Nachrichtenfluss bei UNSUBS-Nachrichten

Eine UNSUBS Nachricht wird immer vom DataClient initiiert. Der Sinn der Nachricht ist es, sich von einer registrierten Prozessvariablenänderung, die durch eine SUBS Nachricht angemeldet wurde, abzumelden.

1. Der Client schickt die UNSUBS Nachricht an den *Kommunikationspartner*
2. Der *Kommunikationspartner* entfernt seinen internen Eintrag, der das Abonnement des Clients mit dieser Prozessvariablen kennzeichnet. Nur in dem Fall, dass dies der letzte (einzige) DataClient ist, der diese Prozessvariable abonniert hat, leitet der *Kommunikationspartner* die UNSUBS Nachricht an den entsprechenden DataServer weiter.
3. Der DataServer empfängt die UNSUBS Nachricht und entfernt die Variable aus seiner internen Liste der abonnierten Variablen. Er sendet daraufhin keine STATE Nachrichten mehr für Änderungen, die an dieser Prozessvariablen erfolgen. Er antwortet auch nicht auf die UNSUBS Nachricht.

6.4.6.5 Nachrichtenfluss bei STATE-Nachrichten

Eine STATE Nachricht wird immer vom DataServer initiiert. Der Sinn der Nachricht ist es, die Änderung einer Prozessvariablen anzuzeigen. Im Falle einer Änderung eines Variablenwertes versendet der DataServer STATE Nachrichten nur dann, wenn Abonnements für diese Variable vorliegen. Änderungen an Prozessvariablen, für die kein Abonnement vorliegt, werden nicht durch STATE Nachrichten angezeigt.

1. Der DataServer schickt die STATE Nachricht an den *Kommunikationspartner*
2. Der *Kommunikationspartner* leitet die Nachricht an alle DataClients weiter, die sich für diese Prozessvariable durch ein SUBS registriert haben.
3. Für die STATE Nachricht erhält der DataServer keine AntwortNachricht vom *Kommunikationspartner*, ebenso werden keine AntwortNachrichten von den DataClients erzeugt, die eine STATE Nachricht vom *Kommunikationspartner* erhalten.

6.5 Kommunikationsschicht 7a: Austausch von Prozessvariablen

Prozessvariablen sind Variablen, die der Steuerung des Prozesses dienen und für mehrere Teilsysteme des Gesamtsystems Maschine von Bedeutung sind. Sie stellen nicht nur Prozessparameter dar, sondern werden auch verwendet, um Kommandos, Statusänderungen usw. zu kommunizieren. (s. Abschnitt 6.6) Sie unterliegen daher Konventionen z.B. zur Namensgebung. Für jede eingeführte Prozessvariable (PV) ist daher zunächst eine Beschreibung zu erstellen.

6.5.1 Component ID

Normalerweise wird der Zustand von Prozessvariablen im gesamten System an alle Komponenten gemeldet, die diese Variable abonniert haben. Für verschiedene Dienste (z.B. Login oder Statusabfragen) ist es jedoch unumgänglich, ein Gerät oder eine andere Komponente direkt anzusprechen. Es ist daher für jede Komponente ein eindeutiger und projektierbarer Name zu vergeben, die ComponentID. Diese ComponentID darf nur die Zeichen 0-9, a-z und A-Z enthalten.
Alle Variablennamen, die nur für eine Komponente gelten sollen, beginnen mit der ComponentID und einem Unterstrich.

6.5.2 Verbundene Daten

Der Zugriff auf Prozessvariablen erfolgt in der Version 2.0 dieser Spezifikation in der Regel auf Basis von Einzelvariablen, unabhängig wie diese in der Komponente datentechnisch organisiert sind.
In bestimmten Fällen ist diese Sichtweise nicht angemessen, da eine PV nur in Verbindung mit anderen PVs einen Sinn ergibt, und die Konsistenz der Verbindung bei unabhängigem Zugriff nicht gewährleistet ist. Dies ist z.B. der Fall, wenn es sich um Augenblickswerte handelt, die beim nächsten Zugriff nicht mehr gültig sind, wenn die Reihenfolge von Zugriffen von Bedeutung ist oder wenn ein Kommando mit den zugehörigen Befehlsparametern übertragen werden soll. Für diesen Fall werden verbundene Daten verwendet, die in Form von XML-Elementen mit Strukturinformationen übertragen werden.

Es wird ausdrücklich darauf hingewiesen, dass verbundene Daten nur für den oben aufgeführten Verwendungszweck vorgesehen sind. Die Verwendung von verbundenen Daten zu Strukturierungszwecken ist nicht zulässig.

Als Grundlage zur Umsetzung der "verbundenen Daten" dienen die folgenden Regeln:

1. Der Inhalt eines Nachrichtentyps kann nur aus
   (a) einer einfachen Zeichenkette (xs:string) oder
   (b) einem XML-Element mit evtl. weiteren Subelementen bestehen (das sogenannte Strukturelement für verbundene Daten).

2. Die Zeichenkette <VDMAXML_P> darf nicht im Inhalt des Nachrichtentyps verwendet werden, da sie zur Synchronisation der Kommunikation verwendet wird.

3. Ein Strukturelement kann wiederum die folgenden Inhalte besitzen:
   (a) Eine einfache Zeichenkette, wobei der Datentyp über das Attribut "d" im umschließenden XML-Startelement näher eingegrenzt werden kann, aber nicht muss (z.B. d="i2"). Wird kein Attribut 'd' übermittelt, wird der Datentyp als 'string' interpretiert.
   (b) XML-Elemente mit gleichem Elementennamen stellen ein komplettes Array dar (siehe "UserRoles" in der Prozeßvariable "*ComponentID*_UserPerm"). Die Reihenfolge der XML-Elemente legt den Index des jeweiligen Array-Elements fest.
   (c) Unterschiedliche XML-Elemente stellen die Mitglieder der Substruktur dar und unterliegen den Regeln eines Strukturelements.

4. XML-Elemente zur Darstellung von "verbundenen Daten" dürfen keine Attribute besitzen. Die einzige Ausnahme ist das Attribut "d" für das Strukturelement mit einer einfachen Zeichenkette als Inhalt (siehe 3a).

5. Im Falle von Arrays müssen immer alle Elemente übertragen werden.

6. Ein Strukturelement eines Typs enthält immer die gleichen Subelemente.

7. Die Nachrichten GET, PUT, SUBS und UNSUBS können nur für die gesamte Struktur und **nicht** für Einzelelemente ausgeführt werden. Ein Dataserver liefert **immer** die gesamte Struktur in einer STATE-Nachricht.

Beispiele:

A) Zweidimensionale Angabe einer Position für einen Sensor:

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

B) Prozessvariable "*ComponentID*_UserPerm" für die Komponente “Panel1”:

```
<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>
```

6.6 Kommunikationsschicht 7b: Definition besonderer Prozessvariablen und Nachrichten-Inhalte

Neben Prozessparametern können mit VDMAXML_P auch Kommandos und Statusänderungen kommuniziert werden. Dazu dienen die im Attribut "s" spezifizierten besonderen Speicherklassen sowie besondere Prozessvariable zur Steuerung der DataServer. Diese besonderen Prozessvariablen sind ggf. gesondert zu vereinbaren.

6.6.1 Besondere Speicherklassen zur Übermittlung von Kommandos und Statusänderungen – Attribut 's'

Das Attribut „s“ (storage class) einer VDMAXML_P-Nachricht beeinflusst das Routing. Damit kann im Datenfluss z.B. zwischen echten Prozessvariablen und z.B. Kommandos unterschieden werden. Insbesondere wird dadurch festgelegt, ob eine Nachricht vom *Kommunikationspartner* der Komponente gespeichert werden soll und ob sie an DataServer weitergegeben wird.

Name	**Bedeutung**
cmd	Kommando, z.B. von der Visualisierung an einen anderen DataClient. Diese Nachrichten sollen im System nicht gespeichert werden (Ereignis-Nachricht)
cur	Augenblickswerte aus dem Prozess, Einstellwerte
fmt	PV wie im Format abgelegt
login	Nachrichten, die ausschließlich an den Login-Server gerichtet sind. (Ereignis-Nachricht)
mem	PV wenn sie vorübergehend extern gespeichert wird
msg	Message; Nachricht wird nicht vom *Kommunikationspartner* gespeichert
state	Statusmeldungen, z.B. wenn ein Kommando ausgeführt wurde
vds	Steuerung der DataServer
visu	Private Information der Visualisierungssysteme

Ereignis-Nachrichten sind solche, die nur im Augenblick ihrer Entstehung gültig sind. In einer Brokerarchitektur werden diese vom Nachrichtenbroker nur einmal an die aktuellen Abonnenten verteilt. Danach kann die enthaltene Information nicht mehr abgefragt werden. Ein Client, der die entsprechende PV erst nach ihrer Verteilung abonniert, erhält demnach erst die nächste Sendung. Alle anderen PVs können jederzeit auf ihren aktuellen Inhalt abgefragt werden.

6.6.2 Interne Variablen zur Steuerung der DataServer und zu Diagnosezwecken

Zur Steuerung des DataServers sowie zu Diagnosezwecken werden folgende Variablenbezeichnungen reserviert:
*ComponentID*_StartServer
*ComponentID*_StartScan
*ComponentID*_ServerStatus
*ComponentID*_ServerMessage

Der Inhalt dieser Variablen wird in einer späteren Version dieses Protokolls festgelegt.

Die Speicherklasse für interne Variable ist ***vds,*** dadurch kann der *Kommunikationspartner* der Komponente sicherstellen, dass die sendende Komponente die Berechtigung zur Steuerung der Datenserver hat. Der globale Variablennamen setzt sich zusammen aus der ComponentID des DataServers und der o.g. Variablenbezeichnung, z.B:
"vds ComponentID_ StartServer".
Falls ein DataServer diese internen Variablen nicht bedienen kann, so darf er sich von Zugriffen darauf nicht beeinflussen lassen.
DataServer können nach Belieben weitere interne Variable anbieten, z.B. Fehlerzähler oder erweiterte Statusanzeigen.

Beispiel:

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

6.6.3 Besondere Prozessvariablen zur Implementierung zentraler Dienste

Die Implementierung zentraler Dienste setzt die Vereinbarung besonderer Prozessvariablen voraus. Diese Fachverbandsschrift enthält im Anhang III (informativ) beispielhafte Realisierungen folgender Dienste:

- Dienst Login/Logout
- Dienst Passwortwechsel
- Dienst Format
- Dienst AuditTrail
- Dienst EventLog

Die darin spezifizierten besonderen Prozessvariablen gehören nicht zum normativen Teil der Spezifikation des Kommunikationsprotokolls VDMAXML_P. Sie sind ggf. gesondert zu vereinbaren.

6.7 Kommunikationsschicht 7c: Dienste

Das Kommunikationsprotokoll VDMAXML_P kann dazu genutzt werden, Dienste zu implementieren, die eine zentrale Benutzerverwaltung, ein zentrales Audit-Trail, eine zentrale Formatverwaltung usw. unterstützen. Als Voraussetzung zur Implementierung derartiger Dienste sind besondere Prozessvariablen zu vereinbaren.

Diese Fachverbandsschrift enthält im Anhang III (informativ) beispielhafte Realisierungen derartiger Dienste auf der Grundlage des Kommunikationsprotokolls VDMAXML_P. Im Einzelnen werden folgende Dienste spezifiziert:

- Dienst Login/Logout
- Dienst Passwortwechsel
- Dienst Format
- Dienst AuditTrail
- Dienst EventLog

Die Spezifikation dieser Dienste gehört nicht zum normativen Teil der Spezifikation von VDMAXML_P. Unterstützt eine Komponente das Kommunikationsprotokoll VDMAXML_P impliziert dies nicht, dass die Komponente auch die in Anhang III spezifizierten Dienste unterstützt. Hierzu bedarf es einer gesonderten Vereinbarung. Es wird daher empfohlen, dass bei Komponenten, die das Protokoll VDMAXML_P unterstützen, zusätzlich anführt wird, welche der in Anhang III spezifizierten Dienste unterstützt werden.

Anhang I (normativ)
In VDMA_XML-Nachrichten verwendete Datentypen

Dieser Abschnitt zeigt die wesentlichen in VDMAXML_P-Nachrichten verwendeten Datentypen. Es müssen nicht alle Datentypen implementiert sein. Die folgenden Datentypen beruhen zum grossen Teil auf der XML-Data Spezifikation.
(http://www.w3.org/TR/1998/NOTE-XML-data)

Name	**Parse Type**	**Speichertyp**	**Beispiele**
number	Eine Zahl mit beliebig vielen Stellen, optional mit Nachkommastellen, optional mit Vorzeichen, optional mit Exponenten. Die Interpunktion erfolgt US-englisch.	string	15 3.1415 123.456E+10
int	Eine ganze Zahl, mit oder ohne Vorzeichen	32 bit Integer	1 -12345
float	wie *number*	64-bit IEEE 488	0.31415 79E+1
fixed.14.4	wie *number* aber mit nicht mehr als 14 Stellen vor, und 4 Stellen hinter dem Dezimalpunkt	64-bit Integer	12.345
boolean	"1" oder "0"	Bit	
dateTime.iso8601	Datum im ISO-8601-Format. Optionale Zeit, Sekundenbruchteile können in Nanosekunden ausgedrückt werden.		2004-03-08T08:15Z (Anm. 1)
dateTime.iso8601tz	Datum im ISO-8601-Format. Optionale Zeit, Sekundenbruchteile können in Nanosekunden ausgedrückt werden.		2004-03-08T08:15:05Z (Anm. 1)
date.iso8601	Datum im ISO-8601-Format. Ohne Zeit.		2004-03-08 (Anm. 1)
time.iso8601	Zeit im ISO-8601-Format. Ohne Datum.		08:15:05Z (Anm. 1)
i1	ganze Zahl, opt. mit Vorzeichen	8 bit Integer	0, -128, +23, 45
i2	ganze Zahl, opt. mit Vorzeichen	16 bit Integer	12, +465, -32768
i4	ganze Zahl, opt. mit Vorzeichen	32 bit Integer	
i8	ganze Zahl, opt. mit Vorzeichen	64 bit Integer	
ui1	ganze Zahl ohne Vorzeichen	8 bit unsigned	
ui2	ganze Zahl ohne Vorzeichen	16 bit unsigned	
ui4	ganze Zahl ohne Vorzeichen	32 bit unsigned	
ui8	ganze Zahl ohne Vorzeichen	64 bit unsigned	
r4	Wie *number*	IEEE 488 4 bytes Float	

r8	Wie *number*	IEEE 488 8 bytes Float	
float.IEEE.754.32	Wie *number*	IEEE 754 4 bytes Float	
float.IEEE.754.64	Wie *number*	IEEE 754 8 bytes Float	
uuid	hexadezimale Ziffern, die Oktette darstellen; optional eingebettete Trennstriche sollen ignoriert werden	128-bytes Unix UUID structure	F04DA480-65B9-11d1-A29F-00AA00C14882
uri	Universal Resource Identifier	Nach W3C-Spezifikation	
bin.hex	hexadezimale Ziffern, die Oktette darstellen	keine vorgege bene Grösse	
char	String	1 Zeichen (UTF-8)	
char.ansi	String	1 ASCII - Zeichen	
string	Pcdata	string (UTF-8)	αβγδεζη
string.ansi	String, der nur ASCII-Zeichen (<=255) enthält.	(8 oder 16 bit pro Zeichen)	Dies ist ein einfacher Text.
string.ansi.long	Sehr langer ASCII-Text.		Dies ist ein Text von ggf. einigen MBytes Länge....
string.long	Sehr langer Unicode-Text.	string (UTF-8)	Dies ist ein Text von ggf. einigen MBytes Länge....
timestamp	Zeitstempel		
timestamp.unix	Zeitstempel nach UNIX-Konvention	32 bit Integer	978618766 (= 4. Jan 2001, 15:32:46)

Kein angeschlossenes Gerät oder DataServer darf sich durch unbekannte Datentypen stören lassen. Im Zweifel kann eine Fehlermeldung ausgegeben werden bzw. in einem Protokollfile vermerkt werden.

Anmerkungen:
1) Stunden, Minuten und Sekunden in Zeitangaben sind immer zweistellig, ggf. mit führenden Nullen, anzugeben. Stunden werden im 24-Stunden-System angegeben. Alle Zeitangaben beziehen sich auf die aktuelle Zeitzone. Sollen Zeiten über Zeitzonen hinweg angegeben werden ist der Buchstabe „Z“ an die Zeitangabe anzuhängen und die Uhrzeit muss auf UTC (Universal Time Coordinated) umgerechnet werden.

Anhang II (normativ)
Textkonventionen

Zeichen Codierung (Character Encoding)
Für die Kommunikation mittels VDMAXML_P wird ein Zeichenencoding entsprechend des UTF-8 Standards festgelegt. Diese Codierung funktioniert sicher auf allen Dateisystemen und ist kompatibel mit allen bisherigen Systemen unter der Voraussetzung, dass nur der US-ASCII Zeichensatz (dezimal 1 bis 127) verwendet wird. Zeichen mit einem Code>127 dienen der Codierung aller anderen UNICODE-Zeichen mit unterschiedlich vielen Bytes/Zeichen. Programmdateien, Konfigurationsdaten usw. werden grundsätzlich als ASCII-Dateien erwartet.

Anmerkung:
UTF-8 und UNICODE sind Vereinbarungen zur Codierung von Zeichen. Ob ein Zeichen wirklich dargestellt werden kann, hängt jedoch davon ab, ob ein entsprechendes Programm benutzt wird und ein entsprechender Font vorhanden ist. Im Zweifel müssen die benötigten Zeichensätze zwischen Endkunde, Hersteller und Zulieferer vereinbart werden.

XML-Dokumente
Die Kommunikation zwischen dem DataServer der Komponente und deren *Kommunikationspartner* auf der der übergeordneten Maschine erfolgt über XML-Dokumente. Dies sind lesbare Texte, die bestimmten Konventionen folgen müssen, damit sie störungsfrei übertragen werden können. Die folgenden Konventionen beziehen sich auf verschiedene Arten von Texten bzw. Dokumenten. Die Struktur der Dokumente ist in XML-Schemata (W3C) festgelegt. XML-Dokumente dürfen in unterschiedlichen Komponenten (Tagnames, Attribute, Inhalte) nur jeweils einen bestimmten Umfang von Zeichen enthalten. Die XML-Spezifikationen wurden für das hier spezifizierte XML-Protokoll für den Datenaustausch zwischen Komponente und Maschine weiter eingeschränkt, um die Verarbeitung der XML-Dokumente zu vereinfachen. Die unten genannten Einschränkungen gelten auch dann, wenn die XML-Spezifikation anderes zulässt.

Tab. AII.1: VDMAXML_P: zusätzliche Einschränkungen für XML-Dokumente

Wo	**Einschränkungen**
Tag-Namen	Dürfen nicht mit "xml" (Groß- und Kleinbuchstaben) beginnen. 1. Zeichen muss ein US-ASCII Buchstabe sein 2.bis n. Zeichen: US-ASCII Buchstaben, Ziffern, Unterstrich
Attribut- Namen	1. Zeichen muss ein US-ASCII Buchstabe sein 2.bis n. Zeichen: US-ASCII Buchstaben, Ziffern, Unterstrich, Punkt, Komma
Attribut-Inhalte	Inhalte, die ein **"** enthalten müssen in **'** eingeschlossen sein. Inhalte, die ein **'** enthalten müssen in **"** eingeschlossen sein. Für **'** und **"** in Attribut-Inhalten können stattdessen die Character-Entities **'** und **"** verwendet werden. Die Zeichen **&**, **<** und **>** müssen durch ihre Character-Entities **&**, **<** und **>** ersetzt werden. Line-feed, Carriage-return und Tabs müssen durch ihre numerische Zeichenreferenzen (**

** und **	**)dargestellt werden |

Tab. AII.2: Namen von Prozessvariablen:

Wo	**Einschränkungen**
Daten-Inhalte	Namen von Prozessvariablen dürfen keine in XML verbotenen Zeichen enthalten.

Anhang III (informativ)
Beispielhafte Realisierung von Diensten auf der Grundlage des Kommunikationsprotokolls VDMAXML_P

Alle definierten Dienste sind prinzipiell als DataServer ausgeführt. Um diese ansprechen zu können, muss die ansprechende Komponente ein DataClient sein. Ist ein Dienst nicht in der Lage, seine geforderte Funktionalität zu erbringen (z.B. weil Datenbankanbindungen nicht vorhanden sind), beendet er seinen ServerSocket und entfernt sich somit aus dem System. Dadurch ist eindeutig gekennzeichnet, dass der Dienst nicht ansprechbar ist. Ein Dienst sollte so implementiert sein, dass die Ausfallzeit möglichst gering ist und er so schnell wie möglich wieder am System teilnehmen kann.

Es wird empfohlen, sich bei der Implementierung der Dienste bei der Reihenfolge der Übertragung der Daten der jeweils spezifizierten Variablen an die Reihenfolge in den Implementierungsbeispielen zu orientieren.

Dienst Login/Logout

Der LoginDienst stellt eine DataServer Komponente dar. Er stellt die Prozessvariablen „LoginRequest“ und „LogoutRequest“ bereit, die bei einem Login bzw. Logout von einem Client des Systems benutzt werden können. Als Parameter wird der Variablen „LoginRequest“ der Loginname, das ggf. verschlüsselte Passwort und die ComponentID des DataClients übermittelt, an dem der Login stattfindet. Bei der Prozessvariablen „LogoutRequest“ ist nur die ComponentID nötig. Es wird der aktuell an diesem Client angemeldete Benutzer ausgeloggt.
Beim Start des LoginServers sind alle abonnierten UserPerm-Variablen auf <LoginResult>Logout</LoginResult>, alle Rollen leer und Expire auf 0 zu setzen.
Die Login Komponente ist Teil des Benutzermanagements des Systems.

Wird aus Sicherheitsgründen das Passwort nur in verschlüsserter Form übertragen, sollte ein asymmetrisches Verschlüsselungsverfahren zum Einsatz kommen. Komponenten, an denen ein Login möglich ist, sollten mindestens das RSA-Verschlüsselungsverfahren unterstützen.[2]

Grundsätzlich hat jede Komponente im System eine ComponentID. Für jede Komponente, die eine eigene Rechtestruktur implementiert, bzw. an einem Login interessiert ist, stellt der LoginDienst eine eigene Prozessvariable „*ComponentID*_UserPerm“ bereit.
Ein Benutzer kann im System gleichzeitig mehrere Rollen einnehmen.

Speicher-klasse (s)	**Name (v)**	**Daten**	**Bedeutung**
login	LoginRequest	ComponentID UserLoginName, Password,	Ein DataClient meldet einen Be-nutzer an.
login	LogoutRequest	ComponentID	Ein DataClient meldet den aktuell angemeldeten Benutzer ab.
cur	PublicKey[3]		Der PublicKey des RSA 1024 Bit Verschlüsselungsverfahrens
cur	*ComponentID*_User Perm	UserLoginName, UserID, UserFullName, LoginResult, UserRoles, Expires	Der Login-Server meldet die Rechte des soeben eingeloggten Benutzers in Form einer State-Nachricht an das System. Mit dem Element UserRoles können ein Rollencode bzw. mehrere dem User zugeordnete Rollen oder ein

[2] Bei einem geplanten Export der Maschine bzw. Komponente in die USA sind die Importrestriktionen der USA für Verschlüsselungsverfahren zu beachten.
[3] Wird nur benötigt, wenn Verschlüsselung zwischen den Kommunikationspartnern vereinbart wird.

			Rollenprofil übermittelt werden. Das Element Expires gibt die Restlaufzeit des Passworts in Tagen an.
cur	*ComponentID_User Name*	UserID, UserLoginName, UserFullName	Dient der Abfrage des aktuell eingeloggten Nutzers

Ablauf Login:

1. Alle Komponenten, die an Logins interessiert sind, registrieren sich durch ein SUBS für ihre entsprechende Prozessvariable „ComponentID_UserPerm“
2. Ein Benutzer loggt sich auf einem DataClient (z.B. Terminal Term1) ein. Der DataClient erfragt dabei vom Benutzer LoginNamen und Passwort.
3. Der DataClient sendet eine PUT Nachricht für die PV „LoginRequest“ des LoginServers an den *Kommunikationspartner*
4. Der *Kommunikationspartner* sendet die Nachricht NUR an den registrierten Server für Logins weiter, da sie von der Speicherklasse „login“ ist
5. Der LoginServer verifiziert den Benutzernamen und das Passwort.
6. Durch die Konfiguration des LoginServers ist festgelegt, welche Komponenten dem Client, von dem der Login durchgeführt wurde, zugeordnet sind und welche lokale Rollen (für die jeweilige Komponente) der eingeloggte Benutzer einnimmt. Aus dieser Information wird für jede dieser Komponenten eine STATE Nachricht für die *ComponentID*_UserPerm Variablen der betroffenen Komponenten generiert und an den *Kommunikationspartner* gesendet. Die Nachrichten enthalten je einen verbundenen Datensatz, bestehend aus:
   a) LoginResult: Logout, Success, LoginRefused, UnknownUser (optional), PasswordInvalid (optional)
   b) UserRole (Name der Rolle) für jede Rolle, die der User einnimmt, sofern Login erfolgreich war
   c) UserFullName sofern Login erfolgreich war
   d) UserID sofern Login erfolgreich war
   e) Expires: Restgültigkeit des Passwords in Tagen (0=abgelaufen, -1=läuft nicht ab)
7. Der *Kommunikationspartner* verteilt die Nachrichten an die entsprechenden Abonnenten der PV „*ComponentID*_UserPerm“
8. Der anfragende DataClient muss die für ihn geltende PV „*ComponentID*_UserPerm abonniert haben. Ein DataClient kann auch so konfiguriert werden, dass er die UserPerm-Variable anderer DataClients (ausschließlich oder zusätzlich) abonniert, wodurch eine Synchronisation der Zugriffsrechte auch über mehrere Terminals und/oder Geräte möglich ist.

Ablauf Logout:

1. Ein Benutzer loggt sich an einem Client aus
2. Der Client sendet die PUT-Nachricht „LogoutRequest“ an den *Kommunikationspartner*. Diese Nachricht enthält die ComponentID des Clients, von dem der aktuell angemeldete Benutzer abgemeldet wird:
3. Der LoginServer setzt für alle dem Client zugeordneten Komponenten die entsprechende Prozessvariable „*ComponentID*_UserPerm“ der Komponente und schickt eine State-Nachricht mit einem verbundenen Datensatz bestehend aus:
   a) LoginResult: Logout
   b) UserRoles leer
   c) UserFullName (optional leer)
   d) UserID (optional leer)
   e) Restgültigkeitsdauer des Passwords : 0
4. Der *Kommunikationspartner* leitet die Meldungen an die für die Meldungen registrierten Komponenten weiter

Details:

Die Speicherklasse „login“ in der Nachricht LoginRequest sorgt dafür, dass die Nachricht vom *Kommunikationspartner* nur ein einziges mal und nur an den LoginServer weitergegeben wird. Sie soll weder vom *Kommunikationspartner* selbst noch von anderen Komponenten gespeichert werden.

Ein Bestandteil der verwendeten Namen ist ComponentID. Dies ist die projektierte Kennung eines DataClient. Sie ist immer dann notwendig, wenn allgemeine Nachrichten nur für einzelne Komponenten bestimmt sind.
Der DataClient oder der LoginServer sollen eine „Event"-Nachricht absenden, die von einem Event-Logger registriert wird. Der LoginServer soll autonom auf falsche Einlogversuche usw. reagieren. Die angeschlossenen Geräte erhalten lediglich die o.g. Informationen.
Für den Logout ist keine besondere Aktion notwendig, da jeder DataClient eine Logout-Nachricht verschicken kann.
Bei einem Login signalisiert nur die Meldung LoginResult=Success eine erfolgreiche Anmeldung. Jede andere Antwort bedeutet, dass der Anmeldeversuch fehlgeschlagen ist. Eine konkrete Implementierung des LoginServers kann an Stelle der Standardmeldung „LoginRefused" konkretere Angaben wie „PasswordInvalid" zurück melden.
Die Rollen und Benutzergruppen müssen projektierbar sein. Jeder DataClient muss die Rollendefinitionen kennen, die für ihn relevant sind. Da die Rollenbezeichnungen nur vom empfangenden Gerät verstanden werden müssen können sie auch Listen oder komplexe Daten (z.B. Profile) enthalten.

Beispiele für Nachrichten:

```
<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>
```

Konfiguration des LoginServers

Die Konfiguration des LoginServers kennt eine Zuordnung von Benutzern zu globalen Benutzergruppen. Eine Benutzergruppe ist eine logische Gruppierung von Benutzern mit gleichen Rechten. Benutzer und Benutzergruppen sind analog zu dem Benutzermanagement eines Windows Systems aufgebaut.
Zusätzlich kennt der LoginServer alle DataClients, an denen sich ein Benutzer aktiv durch einen Login (Angabe von Benutzername und Passwort über eine Bedienoberfläche) anmelden kann. Jedem LoginClient können die Komponenten zugeordnet werden, für die ein entsprechender Login durchgeführt werden soll, wenn sich ein Benutzer an diesem Client anmeldet. Anhand der Zuordnung zwischen dem Benutzer und dem Client, an dem sich der Benutzer angemeldet hat wird für jede dem Client zugeordnete Komponente die komponentenlokalen Rollen für diesen User ermittelt. Diese werden über die „*ComponentID*_UserPerm“ Variable der entsprechenden Komponente als STATE Meldung geschickt. Die Komponente ist dann selbst für die Zurodnung ihrer internen Rechte anhand der übermittelten Rollen verantwortlich.
Jeder DataClient, an dem sich Benutzer anmelden können ist für seinen automatischen Logout selbst verantwortlich. Wird eine konfigurierte Zeit keine Eingabe getätigt, schickt er eine entsprechende Logout Meldung an das System. Alle zugehörigen Komponenten erhalten dann die entsprechende Nachricht, das der Benutzer ausgeloggt wurde.

Dienst Passwortwechsel

Wird aus Sicherheitsgründen das Passwort nur in verschlüsserter Form übertragen, sollte ein asymmetrisches Verschlüsselungsverfahren zum Einsatz kommen. Komponenten, an denen ein Login möglich ist, sollten mindestens das RSA-Verschlüsselungsverfahren unterstützen.[4]

Speicherklasse (s)	**Name (v)**	**Daten**	**Bedeutung**
login	**ChangePasswordRequest**	ComponentID UserLoginName Password	Das Element 'Password' enthält die Unterelemente 'Old' und 'New'
cur	***ComponentID*_ChangePasswordStatus**	ComponentID UserLoginName UserID UserFullName ChangePasswordResult	Die Variable enthält Daten, die für die Darstellung des Ergebnisses des Ersuchs nach einem Passwortwechsel erforderlich sind. Für das Element ChangePasswordResult werden folgende Zustände definiert: Success Refused UnknownUser (optional) PasswordInvalid (optional)

Beispiele für Daten:

```
<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</CComponentID>
			<UserLoginName>ADMIN</UserLoginName>
			<UserID>ADMIN</UserID>
			<UserFullName>Global System Administrator</UserFullName>
			<ChangePasswordResult>Success</ChangePasswordResult>
		</ChangePasswordStatus>
	</STATE>
</VDMAXML_P>
```

[4] Bei einem geplanten Export der Maschine bzw. Komponente in die USA sind die Importrestriktionen der USA für Verschlüsselungsverfahren zu beachten.

Dienst Format

Der Dienst Format stellt eine Dataserver Komponente dar. Diese stellt die unten definierten Prozessvariablen bereit, die für den Datenaustausch zum Zwecke der Formataktivierung und Formatverwaltung erforderlich sind.

Speicher-klasse (s)	Name (v)	Daten	Bedeutung
cmd	SetFormat	FormatID FormatName (optional) FormatVersion FormatStatus	Vom zentralen Formatsystem vorgegebenes Format (Rezeptur) Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'
cur	CurrentFormat	FormatIDFormatID FormatName FormatVersion	Das vorgegebene Format Kann das vorgegebene Format nicht korrekt geladen werden, ist der Inhalt der Subelemente jeweils ein 'Leerstring', also ein leeres Tag.
cmd	SaveFormat	FormatID FormatName FormatVersion FormatStatus	Speichert die aktuellen Prozesswerte als Format ab Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'
cur	Formats	FormatID=0 FormatName=0 FormatVersion=0 FormatStatus	Liefert alle bekannten Formate in einem verbundenen Datensatz zurück Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'

Die Formatverwaltung erfolgt auf Prozessvariablenebene, d.h. alle als formatrelevant gekennzeichneten Prozessvariablen werden beim Laden eines Formats gesetzt, bzw. beim Speichern eines Formats persistent in einer Datenbank abgelegt.

Die Formatverwaltung kennt alle Prozessvariablen, die bei einer Formatumstellung zu berücksichtigen sind. Bei einem Formatwechsel sendet die Formatverwaltung ein PUT für jede Prozessvariable und setzt diese auf den im entsprechenden Format gespeicherten Wert. Anschließend sendet sie ein GET für jede Prozessvariable und überprüft, ob der Wert gesetzt werden konnte. Falls ALLE Prozessvariablen korrekt gesetzt werden konnten, gilt das Format als korrekt gesetzt. Konnte auch nur eine Prozessvariable nicht korrekt gesetzt werden, gilt das Format als nicht korrekt gesetzt. Diesen Status zeigt die Formatverwaltung wiederum über die Prozessvariable „CurrentFormat“ in Form einer STATE Nachricht zur Verfügung.

Eine Komponente, die formatrelevante Prozessvariable besitzt, darf das Setzen der Variablen nicht verzögern, da die Formatverwaltung sonst nicht korrekt prüfen kann, ob der Formatwechsel in Auftrag gegeben wurde oder nicht.

Eine komplexe Komponente, die formatabhängige Konfigurationsdaten benötigt, die sie intern verwaltet, stellt nur eine Prozessvariable für die Aktivierung des Formats zur Verfügung. Wird diese korrekt gesetzt, gilt das Format als übermittelt. Die komplexe Komponente kann anschließend noch mit der internen Formatumstellung beschäftigt sein. Dies kann die Formatverwaltung nicht näher prüfen. Für diesen Fall sollte die Komponente spezielle Prozessvariablen zur Verfügung stellen, die

ein Client entsprechend überprüfen kann. Diese Prozessvariablen sind eine zusätzliche Eigenschaft der Komponente, die ein Client über den normalen Prozessvariablenaustausch erfragen kann. Diese haben keine direkte Verbindung zur Formatverwaltung.

Die Formatverwaltung sollte einen AuditTrail Eintrag vornehmen, der anzeigt, dass ein Format geladen oder gespeichert wurde. Um dies korrekt vornehmen zu können, muss sich die Formatverwaltung beim UserManagement für Login-Prozessvariable registrieren, um den aktuellen Benutzer erfahren zu können. Weiterhin muss sie den AuditTrail Dienst ansprechen, um den Eintrag im AuditTrail vornehmen zu können.

Der AuditTrail Eintrag enthält nur die Information, dass ein entsprechendes Format gesetzt wurde. Die eigentlichen Prozesswerte werden nicht protokolliert, da diese extern abgelegt sind und dort auch protokolliert, bzw. eingesehen werden können.

Vorgehen beim Setzen eines Formats:

1. Ein Client fordert das Setzen eines bestimmten Formats durch ein PUT auf die Prozessvariable „SetFormat“ an, indem er die eindeutige Formatnummer, den Formatnamen (optional), die Formatversion und den Status übergibt
2. Das Formatmanagement liest alle als formatrelevant gekennzeichneten Prozessvariablenwerte für das gewünschte Format und sendet ein PUT für jeden dieser Werte an den *Kommunikationspartner*. Im Falle komplexer Komponenten kann das vorgegebene Format durch ein PUT auf die entsprechende Prozessvariable aktiviert werden.
3. Der *Kommunikationspartner* leitet die PUT Nachrichten an die entsprechenden Komponenten weiter
4. Das Formatmanagement sendet ein GET für jede formatrelevante Prozessvariable um zu überprüfen, ob die Werte korrekt gesetzt werden konnten. Bei komplexen Komponenten erfolgt die Überprüfung durch ein GET auf die entsprechende Prozessvariable.
5. Der *Kommunikationspartner* leitet die GET Meldungen an die entsprechenden Komponenten weiter
6. Die Komponenten antworten mit STATE Nachrichten und liefern dort die aktuellen Werte der Prozessvariablen
7. Der *Kommunikationspartner* der Komponenten leitet die STATE Nachrichten an das Formatmanagement weiter
8. Das Formatmanagement überprüft die STATE Nachricht mit jeder PUT Nachricht. Entsprechen ALLE STATE Meldungen den PUT Meldungen, dann gilt das Format als erfolgreich gesetzt, sonst nicht
9. Das Formatmanagement sendet eine STATE Nachricht der Prozessvariablen „CurrentFormat“ in der angezeigt wird, dass das Format korrekt gesetzt wurde. Kann das vorgegebene Format nicht korrekt geladen werden, ist der Inhalt der Subelemente jeweils ein 'Leerstring', also ein leeres Tag.FormatID

Vorgehen beim Speichern eines Formats:

1. Ein Client fordert das Speichern des aktuellen Formats durch eine PUT Nachricht auf die Prozessvariable „SaveFormat“ an, indem er eine eindeutige FormatID, den Formatnamen (optional), die Formatversion und den Status übergibt
2. Das Formatmanagement fordert alle formatrelevanten Prozesswerte durch entsprechende GET Nachrichten an und speichert die Werte dieser Variablen unter dem gewünschten Formatnamen ab. Anschließend steht das Format zur Verfügung

Vorgehen beim Auslesen aller Formate:

1. Ein Client fordert eine Liste aller Rezepte durch eine GET Nachricht auf die Prozessvariable „Formats“ an.
2. Das Formatmanagement liefert einen verbundnen Datensatz in Form einer STATE Meldung zurück, die alle gespeicherten Formate enthält

Anmerkung Teilformate

Das VDMAXML Protokoll sieht auch Teilformate vor. Diese enthalten nur einen Teilumfang aller formatrelevanten Prozessvariablen. Ein Teilformat wird ansonsten genauso behandelt wie ein Gesamtformat. Wird ein Teilformat geladen, wird ein bereits geladenes Gesamtformat

ungültig.Wird ein neues Gesamtformat geladen, werden alle geladenen Teilformate ungültig.

Zur Unterstützung von Teilformaten muss die Formatverwaltung jedes Format Komponentenorientiert ablegen. Somit ist es möglich, z.B. beim Einrichten nur ein Format für eine bestimmte Komponente zu laden. Für jede Komponente stellt die Formatverwaltung dann die folgenden Prozessvariablen zur Verfügung:

Speicher-klasse (s)	**Name (v)**	**Daten**	**Bedeutung**
cmd	*ComponentID*_SetSubFormat	FormatID FormatName (optional) FormatVersion FormatStatus	Vom zentralen Formatsystem vorgegebenes Format (Rezeptur) für die spezielle Komponente (*ComponentID*) Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'
cur	*ComponentID*_CurrentSubFormat	FormatID FormatName FormatVersion FormatStatus	Das vorgegebene Format für die spezielle Komponente (*ComponentID*) Kann das vorgegebene Format nicht korrekt geladen werden, ist der Inhalt der Subelemente jeweils ein 'Leerstring', also ein leeres Tag.
cmd	*ComponentID*_SaveSubFormat	FormatID FormatName FormatVersion FormatStatus	Speichert die aktuellen Prozesswerte als Format ab Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'
cur	*ComponentID*_SubFormats	FormatID=0 FormatName=0 FormatVersion=0 FormatStatus	Liefert alle bekannten Formate in einem verbundenen Datensatz zurück Für das Element FormatStatus werden folgende Zustände definiert: 'draft', 'released', 'locked' und 'deleted'

Die Prozessvariable *ComponentID*_SetSubFormat wird von einem Client mit PUT gesetzt, um einen Formatwechsel and der durch ComponentID definierten Komponente anzufordern. Die Formatverwaltung liefert auf der Prozessvariable *ComponentID*_CurrentFormat das erfolgreiche oder fehlgeschlagene Setzen in Form einer STATE Nachricht zurück.

So können an einzelnen Komponenten Teilformate gezielt gesetzt werden. Dabei wird jedoch immer das Gesamtformat der Maschine ungültig. Dies muss explizit neu gesetzt werden, wenn ein Gesamtformat geladen werden soll.

Beispiele für Daten:

```
<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>
```

Dienst AuditTrail

Der AuditTrail-Dienst ist ein DataServer. Es bietet die Prozessvariable „AuditTrailMessage“ an, über die AuditTrail Daten geschrieben werden können. Dies erfolgt wie in der Spezifikation beschrieben.
Zur Bestätigung einer geloggten Message sendet der AuditTrail Dienst eine STATE Nachricht für die Prozessvariable „ComponentID_AuditTrailAck“ für die Componente, von der die Message stammt.
Der AuditTrail Server kann mehrere Meldungen in Folge empfangen, ohne diese direkt bestätigen zu müssen.
Über die Prozessvariable „AuditTrailCmd“ wird der AuditTrail Server aufgefordet, eine neue Log-Datei zu beginnen.
Die Sprachkennung erfolgt mittels des Unterelements "Lang". Sie sollte einheitlich nach ISO 639-1 vorgenommen werden. Ergänzungen sollten sich an ISO 3166 orientieren.

Speicherklasse (s)	**Name (v)**	**Daten**	**Bedeutung**
cmd	**AuditTrailCmd**	UserID ComponentID ClientTime	
cur	**AuditTrailMessage**	UserID UserFullName ComponentID MsgID MsgType ClientTime Text Variable (optional)	'MsgID': Eindeutige Kennung einer Nachricht MsgType': Eindeutige Kennung des Typs einer Nachricht 'ClientTime': Unix-Zeitstempel der lokalen Zeit des Clients beim Auslösen der Nachricht 'Text': Inhalt der Nachricht in Textform mit den Unterelementen - TextID - Lang - Entry Hierbei ist die TextID die eindeutige Kennung des Textes, 'Lang' die Sprachkennung und 'Entry' der Texteintrag. 'Variable': Inhalt der Veränderung eines gesetzten Variablenwertes mit den Unterelementen: - Name - Storage - Value Das Unterelement'Value enthält wiederum die Unterelement 'Old' und 'New' mit dem alten bzw. neuen Wert der Variablen.
cur	***ComponentID*_AuditTrailAck**		

Ablauf: Audit-Trail-Meldung
a) Der DataClient sendet eine Audit-Trail-Nachricht

b) Der *Kommunikationspartner* erkennt die Nachricht als Audit-Trail-Nachricht und versieht diese mit einem Zeitstempel im Unix-Zeitformat, um sicher zu stellen, dass die Audit-Trail-Nachricht mit der synchronisierten Systemzeit versehen ist.
c) Der *Kommunikationspartner* leitet die Audit-Trail-Nachricht an den zentralen Audit-Trail-Server weiter.
d) Der AuditTrailServer setzt die Acknowledge-Variable und sendet eine State-Nachricht mit der Variablen ***ComponentID*_AuditTrailAck.**

Beispiel für Daten:

```
<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>
```

Dienst EventLog

Der EventLog Dienst ist ein DataServer, der analog zum AuditTrail Dienst funktioniert. Allerdings gibt es beim EventLog Dienst keine Bestätigung der empfangenen Message in Form einer speziellen Ack-Meldung.

Speicherklasse (s)	**Name (v)**	**Daten**	**Bedeutung**
cmd	**SysEventCmd**	UserID ComponentID ClientTime	
cur	**SysEventMessage**	UserID UserFullName ComponentID MsgID MsgType ClientTime Text Variable (optional)	'MsgID': Eindeutige Kennung einer Nachricht MsgType': Eindeutige Kennung des Typs einer Nachricht ClientTime: Unix-Zeitstempel der lokalen Zeit des Clients beim Auslösen der Nachricht Text: Inhalt der Nachricht in Textform mit den Unterelementen - TextID - Lang - Entry Hierbei ist die TextID die eindeutige Kennung des Textes, 'Lang' die Sprachkennung und 'Entry' der Texteintrag. 'Variable': Inhalt der Veränderung eines gesetzten Variablenwertes mit den Unterelementen: - Name - Storage - Value Das Unterelement'Value enthält wiederum die Unterelement 'Old' und 'New' mit dem alten bzw. neuen Wert der Variablen. Die möglichen Inhalte des Unterelements"Storage" sind durch die für das Attribut 's' definierten Speicherklassen vorgegeben.

Beispiel für Daten:

```
<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>
```

Dienst BatchControl

Um einen Batchwechsel für die Gesamtmaschine konsistent abzuwickeln kann es nötig sein, angeschlossene Geräte in definierter Reihenfolge zu steuern oder abzufragen (z.B. Zählerstände). Diesem Zweck dienen drei Kontrollvariablen, die jedes angeschlossene Gerät bedienen soll, das Batch-relevante Daten erzeugt. Da die Variablen Geräte bezogen sind, beginnen ihre globalen Namen mit der jeweiligen ComponentID.

Vereinbarte PVs

Speicher-klasse (s)	**Name (v)**	**Daten**	**Bedeutung**
cmd	*ComponentID*_BatchCmd		Batch-bezogenes Kommando an die Komponente
cur	*ComponentID*_BatchStatus		Rückmeldung der Komponente über den Status der Batch-relevanten Daten
cur	*ComponentID*_BatchCmdProgress	0-100	Verlauf der Ausführung des BatchCmd (in Prozent)

Kommandos und Statuswerte müssen noch festgelegt werden. Typische Kommandos sind BatchEnd, BatchReset, BatchRestart.

Anhang IV (informativ) Variablennamen

Die nachfolgende Liste enthält eine Zusammenstellung der in diesem Dokument definierten Variablen

Variable	**Bedeutung**	**beschrieben in Abschnitt ...**
*ComponentID*_Start Server	Variable zur Steuerung des DataServers.	6.6.2
*ComponentID*_StartScan	Variable zur Abfrage der aktuellen Einstellwerte	6.6.2
*ComponentID*_ServerStatus	Variable zur Übermittlung von Zustandsmeldungen des DataServers	6.6.2
*ComponentID*_ServerMessage	Nachricht eines DataServers	6.6.2
LoginRequest	Variable enthält die für eine Anmeldung am Login-Server erforderlichen Daten	Anhang III
LogoutRequest	Variable zur Abmeldung am Login-Server	Anhang III
PublicKey	Variable zur Übermittlung des PublicKey des RSA 1024 Bit Verschlüsselungsverfahrens	Anhang III
*ComponentID*_UserPerm	Variable enthält das Login-Ergebnis, die UserID, die Nutzer-Rolle sowie die Restlaufzeit des Passworts	Anhang III
*ComponentID*_UserName	Variable dient zur Abfrage des aktuell an einer Komponente eingeloggten Nutzers	Anhang III
SetFormat	Variable beinhaltet Sollwertvorgabe für Format/Rezeptur	Anhang III
CurrentFormat	Variable beinhaltet die Daten des aktuell eingestellten Formats bzw. der eingestellten Rezeptur	Anhang III
Save Format	Variable zum Speichern der aktuellen Prozesswerte als Format	Anhang III
Formats	Variable zum Auslesen aller bekannten Formate	Anhang III
*ComponentID*_SetSubFormat	Variable beinhaltet Sollwertvorgabe für Format/Rezeptur an der Komponente	Anhang III
*ComponentID*_CurrentSubFormat	Variable beinhaltet die Daten des aktuell eingestellten Formats bzw. der eingestellten Rezeptur	Anhang III
*ComponentID*_SaveSubFormat	Variable zum Speichern der aktuellen Prozesswerte einer komponente als Format	Anhang III
*ComponentID*_SubFormats	Variable zum Auslesen aller bekannten Formate einer Komponente	Anhang III
ChangePasswordRequest	Variable mit Daten, die für einen Passwortwechsel erforderlich sind	Anhang III
*ComponentID*_ChangePasswordStatus	Variable mit Daten, die für die Darstellung des Ergebnisses des Ersuchs nach einem Passwortwechsel erforderlich sind	Anhang III
AuditTrailMessage	Variable, die eine an den Audit-Trail-Server gerichtete Nachricht enthält	Anhang III
AuditTrailCmd	Kommando an den AuditTrail-Server	Anhang III
*ComponentID*_AuditTrailAck	Acknowledge-Variable zur Bestätigung des Erhalts einer AuditTrail-Nachricht	Anhang III
SysEventMessage	Variable mit Inhalt einer Nachricht an den Event-Logger	Anhang III
SysEventCmd	Kommando an den Event-Logger	Anhang III
*ComponentID*_BatchCmd	Batch-bezogenes Kommando an die Komponente	Anhang III
*ComponentID*_BatchStatus	Rückmeldung der Komponente über den Status der Batch-relevanten Daten	Anhang III
*ComponentID*_BatchCmdProgress	Grad der Abarbeitung von ComponentID_BatchCmd in %	Anhang III

Anmerkungen:
Der Namensbestandteil „*ComponentID*_“ ist der eindeutige projektierte Name einer Komponente.

Anhang V (informativ)
Anmerkungen zur Implementierung einer Brokerarchitektur

Der Broker stellt im Konzept des VDMAXML_P nicht nur die zentrale Vermittlungsstelle für Nachrichten dar, sondern realisiert Funktionen, ohne die das Gesamtkonzept schwer zu verstehen ist.
Der Broker kommuniziert gleichzeitig mit 2 Arten von Komponenten: Clients (z.B. die Visualisierung) und DataServers (z.B. eine SPS oder ein angeschlossenes Gerät). Diese beiden Arten unterscheiden sich in der Art der Verantwortlichkeiten und der erlaubten Nachrichtentypen.
Der Broker stellt sich gegenüber den Clients als (TCP-)Server und gegenüber den DataServers als (TCP-)Client dar. Im VDMAXML_P-Konzept gibt es einige Regeln, die das Verhalten des Brokers mit bestimmen:

- Alle Kommunikation bezieht sich auf Prozessvariable (PV)
- Ein Client ist für den Aufbau und die Erhaltung seiner Netzwerkverbindungen verantwortlich.
- Nachrichten haben keinen explizit benannten Empfänger.[5]
- Alle Prozessvariablen sind projektiert und dem Broker bekannt.[6]
- Eine dynamische Änderung von Eigenschaften einer PV in einem DataServer ist nicht vorgesehen.
- Der Variablenhaushalt eines Servers ist projektiert und fix.[7]

Kommunikation mit Clients:
Der Broker akzeptiert Verbindungswünsche von Clients auf einem ServerPort und verarbeitet deren Anfragen.
Optional verlangt der Broker eine Authentifizierung des Clients. (21CFR11)[8]
Der Broker akzeptiert PUT-, GET-, SUBS- und UNSUBS-Nachrichten von den Clients.
PUT, GET und SUBS-Nachrichten werden an die betroffenen DataServer, nicht aber an Clients weitergegeben.
Eine UNSUBS-Nachricht wird nur dann an einen DataServer gesendet wenn es keine weiteren Subscriber für die angesprochene Variable mehr gibt.
Wird eine Clientverbindung unterbrochen, werden dessen Abonnements intern gekündigt.
Jede PUT, GET und SUBS-Nachricht für eine Variable, die entweder nicht bekannt ist oder deren Quelle z.Zt. nicht verfügbar ist wird mit einer „Invalidate“-Nachricht beantwortet.[9]
Die Prozess-globalen Namen der Variablen werden bei der Weitergabe an die Server in deren interne Item-Bezeichnungen umgesetzt.

Kommunikation mit Servern:
Der Broker verbindet sich automatisch mit allen im System bekannten DataServers
Von Servern akzeptiert der Broker nur STATE-Nachrichten. Diese werden an alle Abonnenten für die jeweilige Variable weiter geleitet.
Bei PUT, GET und SUBS-Nachrichten der Clients werden die DataServer entsprechend des RWU-Attributs der PV-Beschreibung für den jeweiligen Server angesprochen.[10]

---

[5] Clients erhalten eine Nachricht wenn sie die angesprochene Variable abonniert haben, Server erhalten sie wenn die Variable in ihrem Variablenhaushalt projektiert ist. Ein Sender kann nicht bestimmen, an wen die Nachricht gehen soll.
[6] Das gilt für die Speicherklassen „cur“, „fmt“ und „mem“. Andere Speicherklassen werden von festgelegten Servern (login) oder projektierten Servern bedient.
[7] Das gilt für die Speicherklassen „cur“, „fmt“ und „mem“. Andere Speicherklassen werden von festgelegten Servern (login) oder projektierten Servern bedient.
[8] Der Ablauf muss noch definiert werden. Das Feature ist Protokoll-kompatibel realisierbar. Die Realisierung könnte der nächsten Version des Protokolls erfolgen.
[9] Es könnten hier noch vereinbarte Fehlercodes zurück geliefert werden.
[10] Jede PV muss für jeden DataServer projektiert sein. Minimal notwendig sind Angaben über den Server-internen Variablennamen (Item) und die Schreib/Lese-Fähigkeit (RWU-Attribut). Ist eine PV für mehr als einen Server projektiert, darf nur ein Server die Variable liefern (Quelle, readable). Es ergibt sich aus dem Protokoll, dass nur PUT-Nachrichten an mehr als einen Server geschickt werden, eine „Querverbindung“ zwischen 2 Servern entsteht dadurch also nicht, da Server keine PUT-Nachrichten senden dürfen. Die Konfigurationsdateien sollten standardisiert sein, damit ein einfacher Austausch möglich ist.

Beim Zusammenbruch der Kommunikation mit einem Server

- erzeugt der Broker eine „invalidate“-Nachricht für jede Prozessvariable, die der verlorene Server hätte bedienen müssen (nur Subscriptions).
- versucht der Broker zyklisch, die Kommunikation mit dem Server wieder herzustellen.
- erzeugt er einen Eintrag in eine Log-Datei (limitierte Anzahl)

Nach Wiederherstellung der Verbindung werden alle anstehenden Abonnements bei dem DataServer erneuert.
Während der Unterbrechung werden weiter Abonnements für die betroffenen PVs akzeptiert. Sie werden jedoch mit einer initialen „invalidate“-Nachricht beantwortet.
Die Geräte-internen Item-Bezeichnungen der Variablen werden bei der Weitergabe an die Clients in die Prozess-globalen Namen umgesetzt.

Routing
Das Standard-Routing für die Speicherklassen „cur“, „fmt“ und „mem“ wird aus den Konfigurationsdaten der DataServer abgeleitet.
PVs mit der Speicherklasse „cmd“ werden nur einmalig an die Server weitergegeben. Sie werden nicht gespeichert.[11] [12]
PVs mit der Speicherklasse „login“ werden nur an den konfigurierten Login-Server weiter gegeben.
Für alle anderen Speicherklassen wird ein DataServer explizit projektiert. Alle Nachrichten dieser Klassen werden dann nur von diesen Servern bedient.[13]

Konfigurationsdaten:
Der Broker benötigt folgende Konfigurationsdaten:

- Den eigenen ServerPort für Client-Anfragen
- Die Namen und IP:Port-Adressen aller DataServer
- Eine Variablenbeschreibungsdatei für jeden DataServer mit Angaben zu Name-Item-Relation und Schreib/Lese-Fähigkeit[14]
- Die Namen und IP:Port-Adressen aller DataServer für bestimmte Speicherklassen.

[11] Nachrichten von der Speicherklasse „cmd“ sind Kommandos, die von einem Client geschickt werden. Sie werden an diejenigen Server geliefert, die diese PV „writable“ projektiert haben. Kommandos sind transiente Ereignisse, Es ist nicht erwünscht, dass eine später hinzu kommende Komponente ein vergangenes Kommando ausführt. Eine Speicherung eines Kommandos ist also weder notwendig noch sinnvoll.
[12] Die klare Trennung von Server und Client bezüglich der unterstützten Nachrichtentypen bedeutet, dass der Empfänger eines Kommandos ein Server sein muss. Da Server-Variablen projektiert sind, ist auch sicher gestellt, dass ein Kommando nur an Komponenten weitergeleitet wird, die zur Ausführung des Kommandos autorisiert sind.
[13] Das Broker-Konzept eignet sich gut zum Austausch dynamisch wechselnder Zustände z.B. zwischen zwei Visualisierungsgeräten. Dazu werden Pseudo-Prozessvariable z.B. in der Speicherklasse „visu“ eingerichtet. Da diese im realen Prozess keine Bedeutung haben ist ihre Projektierung ein ebenso großer wie überflüssiger Aufwand. Einfacher ist es, alle Nachrichten von einer Speicherklasse wie „visu“ an einen einzigen DataServer weiterzugeben, der die Variablen dynamisch anlegen und den Abonnenten zur Verfügung stellen kann.
[14] Jede PV muss für jeden DataServer projektiert sein. Minimal notwendig sind Angaben über den Server-internen Variablennamen (Item) und die Schreib/Lese-Fähigkeit (RWU-Attribut). Ist eine PV für mehr als einen Server projektiert, darf nur ein Server die Variable liefern (Quelle, readable). Es ergibt sich aus dem Protokoll, dass nur PUT-Nachrichten an mehr als einen Server geschickt werden, eine „Querverbindung“ zwischen 2 Servern entsteht dadurch also nicht, da Server keine PUT-Nachrichten senden dürfen. Die Konfigurationsdateien sollten standardisiert sein, damit ein einfacher Austausch möglich ist. Jede PV muss für jeden DataServer projektiert sein. Minimal notwendig sind Angaben über den Server-internen Variablennamen (Item) und die Schreib/Lese-Fähigkeit (RWU-Attribut). Ist eine PV für mehr als einen Server projektiert, darf nur ein Server die Variable liefern (Quelle, readable). Es ergibt sich aus dem Protokoll, dass nur PUT-Nachrichten an mehr als einen Server geschickt werden, eine „Querverbindung“ zwischen 2 Servern entsteht dadurch also nicht, da Server keine PUT-Nachrichten senden dürfen. Die Konfigurationsdateien sollten standardisiert sein, damit ein einfacher Austausch möglich ist.