Software-Dokumentation

„PMS_Mod.dll“ - Ankopplung an ScalarVis und Wonderware

erstellt von Klaus Raithel

Version 24 (18.06.04)
(in Arbeit)

KoSA

COMPUTER OUTFIT

Inhalt

Abbildungen

Tabellen

1 Allgemeines

Da bei KoSa verschiedene Datenerfassungssysteme (**ScalarVis** und **Wonderware**[1]) im Einsatz sind, deren Daten an alle PMS-Programmen weitergereicht werden müssen, es andererseits aber auch notwendig ist, bei Störungen zwischen Redundanzsystemen zu wechseln, musste eine allgemeine Schnittstelle geschaffen werden, die diese Aktivitäten möglichst transparent durchführt.

Da die Alpha-Rechner, auf denen **ScalarVis VMS** läuft, nicht mehr supported werden, ist es kurz- bis mittelfristig geplant, diese Hardware außer Betrieb zu nehmen. Allerdings ist die Implementierung von ScalarVis auf NT/W2K nicht sehr zuverlässig und auch zu teuer, weshalb nach Alternativen Ausschau gehalten wurde.

Auf der anderen Seite musste aber trotzdem die Entwicklung des Frontends ungeachtet des darunter liegenden Systems durchgeführt werden. Daher empfahl es sich, die gesamte Bedienung dieser Schnittstellen zu kapseln und zu virtualisieren, mit dem Ziel, die Frontend-Software von den Einzelheiten der Datenabfrage zu befreien. Dazu wurde das Modul **PMS_Mod.dll** geschrieben.

1.1 Eingesetzte Software und deren Eigenschaften

1.1.1 ScalarVis

Zur Übertragung der Messdaten von der Anlage (z.B. Poly II Konti) zur Client-Software wird zurzeit noch die Software **„ScalarVis VMS“** (installiert auf einer Alpha) und **„ScalarVis NT“** (installiert auf Win2000) der Firma InfraServ verwendet. ScalarVis stellt seinen Klienten die Daten in Form von Strukturen, die hier als „Tags“ bezeichnet werden, zur Verfügung. Ein Tag enthält üblicherweise die Daten eines Messpunktes, jedoch ist es auch möglich, Konstanten oder Berechnungen – die wiederum die Werte anderer Tags beinhalten können – zu hinterlegen. Aus datentechnischer Sicht stellt sich ein Tag einfach als eine Struktur mit mehreren Elementen dar.

Zur Kommunikation zwischen **ScalarVis** und der Frontend-Software existiert das ActiveX-Modul **MESSVB1.OCX**, das in ein VisualBasic-Fenster eingebunden wird und dann über Eigenschaften und Methoden direkt angesprochen werden kann. **MESSVB1.OCX** kann auf einem anderen Computer als **ScalarVis** laufen, jedoch muss **ScalarVis** selbstverständlich über das Netzwerk erreichbar sein.

Die Kommunikation läuft prinzipiell so ab, dass der Client **ScalarVis** mitteilt, von welchem Tag er Daten erhalten möchte und danach die einzelnen Eigenschaften aus globalen Eigenschaften liest. Speziell dann, wenn die kompletten Informationen zu den Tags ermittelt werden müssen, ist dieser Vorgang relativ langsam. Er lässt sich jedoch optimieren, indem die allgemeinen Daten (Name, Grenzwerte, Rezeptwerte, etc.) nur einmal gelesen und zwischengespeichert werden. Bei einer normalen Abfrage werden dann nur die Live-Daten gelesen, wodurch der Vorgang erheblich beschleunigt wird.

Ein Problem ist, dass die Implementierung auf VMS zwar hervorragend funktioniert, die Windows-Adaption aber einige Teile nicht oder nur sehr umständlich unterstützt, so dass einige „interessante“ Lösungen gefunden werden mussten.

[1] Außerdem war noch die Software „PI“ im Gespräch, jedoch konnte sich diese Lösung nicht qualifizieren und wird daher nicht weiter unterstützt.

1.1.2 Wonderware

Wonderware verwendet einen völlig anderen Ansatz. Prinzipiell stellt Wonderware einen OLEDB-Provider (Industrial SQL) zur Verfügung, der dann über einen Microsoft SQL-Server angesprochen wird. Durch dieses Schema lassen sich normale SQL-Befehle zur Abfrage und Steuerung verwenden, was die Implementation stark vereinfacht.

Trotz der zusätzlichen Schicht des SQL-Servers liefert Wonderware die Daten in sehr guter Geschwindigkeit, wodurch auch schnelle Änderungen gut aufgelöst werden können.

Da Wonderware auch ein gutes Visualisierungsmodul mitliefert, werden in Zukunft die Anlagen nach und nach darauf umgestellt.

1.2 Implementierung

Das ActiveX-Modul **PMS_Mod** wurde intern in zwei Schichten realisiert. Die untere Schicht (repräsentiert durch die Klassen **clsMessdas** und **clsWWare**) kommuniziert direkt mit ScalarVis, bzw. Wonderware, ist aber von außen nicht sichtbar. Die obere Schicht (die Klasse **PMSKopplung**) ist dagegen sichtbar und wird von der Client-Software direkt angesprochen. Sie dient hauptsächlich dazu, die vom Client gestellten Anfragen und Kommandos an die untere Schicht weiterzuleiten und die Ergebnisse nach oben zu transportieren.

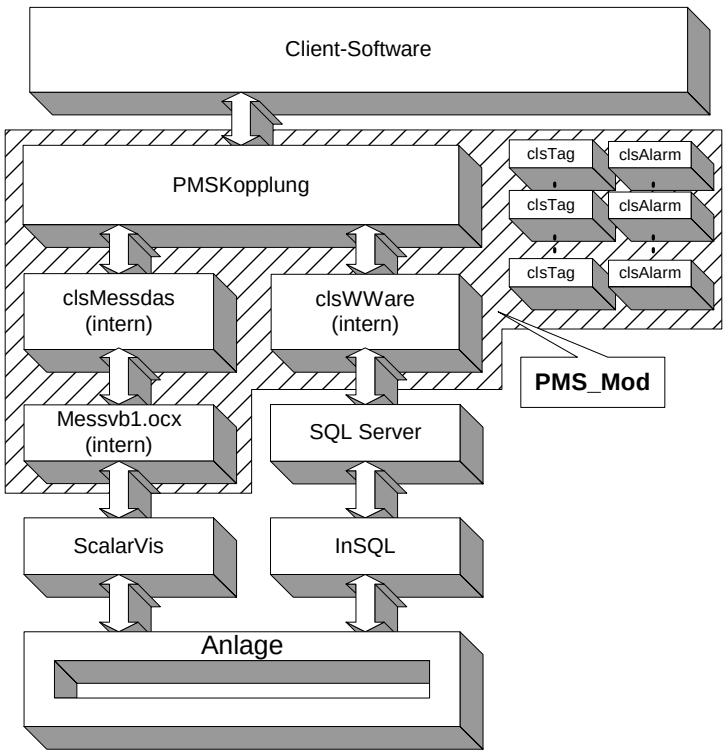

Abbildung 1: Struktur von PMS_Mod

PMSKopplung sowie die untergeordneten internen Klassen verwenden zwei weitere öffentliche Klassen, **clsTag** und **clsAlarm**, in denen die einzelnen Daten der Tags, bzw. Alarm-Ereignisse gespeichert werden. Diese beiden Klassen sind allgemein gehalten, damit sie sowohl mit **ScalarVis**, als auch mit **Wonderware** verwendet werden können. Daher muss beim Wechsel zwischen den Systemen nur das entsprechende System angegeben werden, die restliche Anpassung übernimmt **PMSKopplung** durch die Auswahl der passenden internen Schnittstelle.

2 Voraussetzungen und Installation

2.1 Betriebssystem

Als Betriebssystem wird Windows NT 4 (mindestens SP5) oder Windows 2000 (SP2) vorausgesetzt.

2.2 Benötigte Dateien, etc.

Zur Kommunikation mit ScalarVis muss **MESSVB1.OCX** installiert sein. Dadurch werden auch – sofern noch nicht vorhanden – die Dateien **MFC42.DLL** und **MSVCRT.DLL** installiert.

Auch wenn ScalarVis nicht verwendet werden soll, ist es trotzdem notwendig, **MESSVB1.OCX** zu installieren!

Außerdem wird **MDAC 2.7** oder höher benötigt, um auf Datenbanken und Recordsets zugreifen zu können.

Weiterhin muss die Datei **MSVBVM60.DLL** ab Version 6.0 vorhanden sein.

2.3 Installation

Die Installation der Komponente ist einfach und beschränkt sich darauf, die Datei ins System-Verzeichnis (z.B. **C:\WINNT\system32**) zu kopieren und folgendes Kommando auszuführen:

```
C:\WINNT\system32>regsvr32 PMS_Mod.dll
```

Danach erscheint die folgende Meldung:

Abbildung 2: Bestätigung der Registrierung

Alternativ wird das Modul auch mit dem Installationspaket **PMS/Basis** installiert, das auch dafür sorgt, dass die anderen Voraussetzungen gegeben sind.

Nach dieser Installation können die entsprechenden Applikationen mit der DLL arbeiten.

2.4 Verwendung in VB

In Visual Basic den Menüpunkt **„Project/References“** (bzw. **„Projekt/Referenzen“**) öffnen, in der Liste das Modul namens **„PMS_Mod“** suchen und die Checkbox aktivieren. Danach können die Objekte **PMSKopplung**, **clsTag** und **clsAlarm** sowie die Enums im Projekt verwendet werden (siehe dazu **PMSKopplung**/Vorgehensweise).

3 Die einzelnen Komponenten

In der DLL sind die folgenden öffentlichen Klassen, resp. COM-Objekte definiert, die zusammen die Gesamtfunktionalität ergeben:

- **PMSKopplung**
- **clsTag**
- **clsAlarm**

Weiterhin sind noch die folgenden Enums definiert:

- **pmsRefreshType**
- **pmsErrorcode**
- **pmsArchiveTypes**
- **pmsCollectorType**
- **tagState**
- **tagAction**
- **Commands**
- **mesFunc**

3.1 PMSKopplung

Die Klasse **PMSKopplung** stellt prinzipiell einen Container dar, der folgende Aufgaben übernimmt:

- Vollständige Kapselung der Anlage von der Applikation.
- An- und Abmeldung an der Anlage.
- Verwaltung der Anmeldedaten – auch für mehrere Anlagen (zur internen Redundanzbildung).
- Überwachung der Anlage auf Ausfälle und – sofern gewünscht – automatisches Umschalten auf alternative Anlagen bzw. Hosts (vollautomatische Redundanz)[2].
- Bereitstellen der Tags in Form von Collections mit aktuellen Werten nach interner oder externer Triggerung.
- Bereitstellen der aktuellen Alarme aus der Anlage, sowie Aufbauen und Bereitstellen einer Alarmhistorie, ebenfalls als Collection[3].
- Senden von Einstellungen (Grenzwerte, Rezepte, etc.) an die Anlage[4].

PMSKopplung arbeitet dabei als möglichst dünne Schicht über den internen Schnittstellenklassen **clsMessdas** und **clsWWare**, die wiederum die proprietären Datentypen, Formate und Methoden der eigentlichen Datenerfassungsmodule in eine allgemein verwendbare Form bringen und die Ergebnisse in den Tags ablegen.

[2] Dabei wird die automatische Redundanz jedoch nicht dazu verwendet, zwischen ScalarVis und Wonderware zu wechseln.

[3] Die Alarme werden zwar noch intern bereitgestellt, aber nicht mehr benötigt, da sie durch **PMS_DbAlarm.exe** über die PMS-Datenbank zur Verfügung gestellt werden.

[4] Zurzeit (April 2004) noch nicht implementiert.

3.2 clsTag

Die Klasse **`clsTag`** beinhaltet alle Informationen zu einem Tag. Dazu gehören hauptsächlich:

- Tag-Referenz (die Nummer des Tags)
- Aktueller Messwert (= Istwert)
- SI-Einheit des Messwerts
- Sollwert
- Grenzwerte (Warn- und SOC-Grenzen)
- Rezept-Soll- und Grenzwerte

Die aus den Tags ermittelten Daten werden in jeweils einer Instanz dieser Klasse abgelegt und zur Applikation transportiert.

Üblicherweise werden die Tags als Elemente einer Collection geliefert.

3.3 clsAlarm

Die Klasse **`clsAlarm`** enthält die Informationen zu jeweils einer Alarmsituation. Dazu gehören unter anderem:

- Alarm-Referenz (Nummer des Alarms)
- Tag-Referenz (die Nummer des Tags, bei dem die Störung auftrat)
- Fehlercode
- Status und Statustext
- Zeitpunkt des Auftretens, bzw. des Verschwinden des Alarms

Auch die Alarme werden üblicherweise in einer Collection geliefert.

Zu beachten ist hierbei, dass *in der Historie* sowohl beim Auftreten, *als auch beim Verschwinden* eines Alarms ein Eintrag vorgenommen wird. Bei der Liste der aktuellen Alarme dagegen wird das Verschwinden eines Alarms nicht dargestellt (da der Alarm ja noch anliegt, würde das keinen Sinn machen).

Die auf diese Weise ermittelten Alarme werden jedoch nicht mehr verwendet; die Alarme werden aktuell mit dem Programm **„`PMS_DbAlarm.exe`“** ermittelt und in der PMS-Datenbank gespeichert.

Es ist daher zu erwägen, ob die Methoden und Funktionalitäten zur Alarmierung nicht entfernt oder zumindest lahm gelegt werden sollen, um die allgemeine Performance weiter zu steigern.

4 Die Klassen im Detail

Hinweis: Alle folgenden Beschreibungen beziehen sich auf Visual Basic 6.0 als Entwicklungsumgebung.

4.1 PMSKopplung

Diese Klasse wird vom Client direkt angesprochen. Je nach Einstellung erzeugt sie wiederum genau eine Instanz der darunter liegenden Klassen **clsMessdas**, **clsWWare** oder **clsPI**[5].

Wird die Einstellung verändert, so wird die bisherige Instanz der Hilfsklasse gelöscht und eine neue Instanz erstellt.

Die meisten Methoden und Eigenschaften werden von **PMSKopplung** direkt an die Hilfsklassen weitergereicht. Die Ausnahme hiervon ist die Eigenschaft **ConnectionString**, da hierbei auch die zu verwendende Hilfsklasse definiert werden kann. Daher wird der übergebene Steuerstring auch innerhalb von **PMSKopplung** schon teilweise analysiert.

4.1.1 Vorgehensweise

Um mit der Klasse arbeiten zu können, wird eine Instanz davon erzeugt:

```
Private WithEvents PMS As PMSKopplung
```

oder

```
Public WithEvents PMS As PMSKopplung
```

Der Name (hier „**PMS**“) kann selbstverständlich frei gewählt werden. Danach wird die Instanz – z.B. im Ereignis **Load** der Form – aktiviert:

```
Set PMS = New PMSKopplung
```

Nach diesen Vorbereitungen muss nun die Anmeldung erfolgen. Bei ScalarVis werden dazu vier Parameter benötigt:

Hostname	Name oder IP-Adresse des Computers, auf dem das ScalarVis läuft
Port	Nummer des TCP/IP-Ports, über den ScalarVis erreichbar ist
Username	Name des Benutzers, der berechtigt ist, Daten aus ScalarVis zu lesen
Kennwort	Kennwort zum Anmelden des Benutzers

Diese Parameter werden alle zusammen über die Eigenschaft **ConnectionString** zugewiesen:

```
PMS.ConnectionString = "Host=Srv01;Port=32768;User=HansDampf;Pwd=MachAuf"
```

Hiermit wird noch keine Verbindung hergestellt, sondern nur die notwendigen Parameter bereitgestellt. Als nächster Schritt folgt die Initialisierung, die dann die tatsächliche Verbindung herstellt:

```
If Not PMS.Init Then
   Fehlerbehandlung
End If
```

[5] Da PI nicht verwendet wird, ist diese Schnittstelle auch nicht aktiv. Sie existiert zwar, liefert aber keinerlei Daten.

Danach können zwar schon Tags geliefert werden, jedoch sind diese noch nicht mit echten Daten gefüllt. Dazu müssen sie jetzt noch „aktualisierend“ gemacht werden:

```
PMS.EnableActualizing 1,20
PMS.EnableActualizing 101,130
```

In diesem Beispiel werden die Tags **1** bis **20** und **101** bis **130** „scharf“ geschaltet. Alle anderen liefern nur leere Datensätze. Diese Aufteilung wurde vorgenommen, um die Leistung der Klasse zu optimieren, da das Abfragen der Daten bei ScalarVis ein relativ langwieriger Prozess ist und es keinen Sinn macht, unnötige, bzw. uninteressante Daten einzufordern.

Danach können die Daten ausgelesen werden. Dies kann auf zwei verschiedene Methoden geschehen:

1. Die Methode **PMS.DoRefresh** aufrufen, z.B. in einer Timer-Routine. Dadurch werden die Daten von der Anlage abgeholt und in der Collection bereitgestellt.

```
Private Sub tmrRefresh_Timer()
   If Not PMS Is Nothing And PMS.ValidData Then PMS.DoRefresh
End Sub
```

2. Mit der Methode **StartInternalTimer** wird ein interner Timer gestartet, der in regelmäßigem Abstand (der aus der jeweiligen **RefreshTime** der Anlage berechnet wird) **DoRefresh** aufruft.

```
PMS.StartInternalTimer
```

   Diese Methode hat den Vorteil, dass in der Applikation kein Timer zur Abfrage der Daten benötigt wird. Wenn eine andere als die vorgegebene Frequenz gewünscht wird, so kann diese an die Methode übergeben werden:

```
PMS.StartInternalTimer 300
```

   In diesem Beispiel wird **DoRefresh** alle fünf Minuten aufgerufen[6].

Die Methode **DoRefresh**, die auf eine dieser beiden Arten aufgerufen wird, liest dann die Daten aus und feuert ein Event **Refresh** mit dem Parameter **pmsRefreshTags**. In der behandelnden Methode können dann die gewünschten Daten ausgelesen werden.

Sofern Alarme aufgetreten sind, feuert die Methode außerdem noch das gleiche Event, jedoch mit dem Parameter **pmsRefreshAlarms**, sowie **pmsRefreshAlarmUpdate**, falls sich die Alarmhistorie geändert hat.

Beispiel der Behandlung eines Events:

[6] Da die normale Timerfunktion nur ein maximales Intervall von ca. 65,5 Sekunden zulässt, wird der interne Timer auf 1 Sekunde gestellt und eine entsprechende Variable heruntergezählt, um auch längere Intervalle realisieren zu können.

```
Private Sub PMS_Refresh(RefreshType As pmsRefreshType)
   Select Case RefreshType
      Case pmsRefreshTags:
         ReadAndDisplayTags
      Case pmsRefreshSend:
         ReadAndDisplayTags
      Case pmsRefreshStatistics:
         ReadAndDisplayTags
      Case pmsRefreshAlarms:
         lngAnzahlAktuellerAlarme = PMS.AlarmCount
      Case pmsRefreshAlarmUpdate:
         lngAnzahlAlarmHistorie = PMS.AlarmHistCount
   End Select
End Sub
```

Die hier beispielhaft angedeutete Prozedur **ReadAndDisplayTags** holt die Collection von Tags aus der Klasse und zeigt die interessanten Werte an.

4.1.2 Funktionen

Im Folgenden werden die öffentlichen Funktionen der Schnittstelle beschrieben. Alle hier beschriebenen Funktionen und Prozeduren sind als **Public** deklariert.

4.1.2.1 Function ChangeServer() As Boolean

Parameter: *Keine*

Rückgabe:

True	Der Server wurde erfolgreich gewechselt
False	Der Server konnte nicht gewechselt werden

Mögliche Fehler/Warnungen:

pmsErrorConnect	Keine Verbindung möglich

Mögliche Events:

Connection	Der Server wurde gewechselt, sein Name wird als Parameter geliefert.

Wenn *mehrere* Verbindungssätze (siehe dazu **ConnectionString**) definiert sind, wechselt diese Methode zum nächsten Verbindungssatz. Existiert dagegen nur *ein* Verbindungssatz, so liefert die Methode immer **False** zurück und ändert nichts an den internen Einstellungen.

Zu beachten ist, dass beim erfolgreichen Wechsel des Servers die Liste der zu aktualisierenden Tags und der zu versendenden Alarme nicht verändert wird. Das bedeutet, dass ein „fliegender Wechsel“ vorgenommen wird und die Werte ab jetzt vom neuen Server geliefert werden.

Dieses Verhalten ist erwünscht und speziell bei redundanten Systemen, bei denen zwei oder mehr Server eine einzige Anlage überwachen, sinnvoll.

4.1.2.2 Function CheckConnection() As Boolean

Parameter: *Keine*

Rückgabe:

True	Verbindung zur Anlage besteht und ist funktionsfähig
False	Keine Verbindung vorhanden

Mögliche Fehler/Warnungen:

pmsErrorConnectionLost	Bisher bestehende Verbindung ist verloren gegangen.

Diese Methode prüft, ob eine Verbindung zur Anlage besteht und funktionsfähig ist. Dazu wird (nur bei bestehender Verbindung) ein Test zur Kommunikation mit der Anlage durchgeführt.

Ein Rückgabewert **`True`** bedeutet daher, dass eine funktionsfähige Verbindung zur Anlage besteht. **`False`** wird dagegen sowohl vor dem Aufbau einer Verbindung, als auch beim Abbruch derselben gemeldet. Die Methode selbst versucht *nicht*, eine Verbindung herzustellen!

Schlägt der Kommunikationstest bei einer bestehenden Verbindung fehl, so feuert die Methode ein Event **`Warning`** mit dem Parameter **`pmsErrorConnectionLost`**. Da diese Methode auch von den internen Funktionen verwendet wird, bedeutet das, dass beim Verlust der Verbindung dieser Event erzeugt wird.

4.1.2.3 Function Connect() As Boolean

Parameter: *Keine*

Rückgabe:

`True`	Eine Verbindung zur Anlage bestand bereits oder wurde hergestellt
`False`	Es konnte keine Verbindung hergestellt werden

Mögliche Fehler/Warnungen:

`pmsErrorCantConnect`	Fehler beim Verbinden
`pmsErrorInvalidParams`	Ungültige Parameter

Mögliche Events:

`Connection`	Eine neue Verbindung wurde hergestellt; der Name des Servers wird als Parameter geliefert.

Diese Methode stellt sicher, dass eine Verbindung zur Anlage besteht. Existiert bereits eine Verbindung, so wird nichts verändert. Ansonsten wird versucht, die Verbindung – mittels der Methode **`ReConnect`** – aufzubauen. Da diese Methode bei **`DoRefresh`** automatisch aufgerufen wird, muss sie üblicherweise nicht explizit aufgerufen werden.

4.1.2.4 Function DisableActualizing() As Boolean

Parameter: *Optional ByVal lTagFirst As Long = -1,*
Optional ByVal lTagLast As Long = -1

`lTagFirst`	Erstes zu deaktivierende Tag; Default = das erste mögliche.
`lTagLast`	Letztes zu deaktivierende Tag; Default = das letzte mögliche.

Rückgabe:

`True`	Aktualisierungsflags wurden erfolgreich gelöscht
`False`	Es ist noch keine Tagliste definiert oder **`lngTagFirst < 1`** oder **`lngTagLast > TagLast`**

Diese Methode deaktiviert die Aktualisierungsflags der angegebenen Tags. Danach werden die Tags bei **`DoRefresh`** nicht mehr aktualisiert, sondern enthalten in allen Double-Feldern den Wert **`-1E+38`**, in den Long-Feldern **`0`** sowie leere Textfelder.

Bei den damit deaktivierten Tags wird auch automatisch die Alarmierung ausgeschaltet (siehe dazu auch **`DisableAlarms`** und **`EnableAlarms`**).

4.1.2.5 Function DisableAlarms() As Boolean

Parameter: *Optional ByVal lTagFirst As Long = -1,*
Optional ByVal lTagLast As Long = -1

lTagFirst	Erstes Tag, das nicht mehr alarmieren soll (Default = erstes aktualisierende Tag)
lTagLast	Letztes Tag, das nicht mehr alarmieren soll (Default = letztes Tag)

Rückgabe:

True	Aktualisierungsflags wurden erfolgreich gelöscht
False	Es ist noch keine Tagliste definiert oder **lngTagFirst** < 1 oder **lngTagLast** > **TagLast**

Mit dieser Methode werden nur die Alarmierungen der angegebenen Tags abgeschaltet. Der Aktualisierungsstatus bleibt erhalten. Dadurch werden diese Tags bei **DoRefresh** zwar noch aktualisiert, jedoch wird von diesen Tags kein Alarm mehr geliefert.

4.1.2.6 Function Disconnect() As Boolean

Parameter: *Keine*

Rückgabe:

True	Verbindung besteht nicht mehr
False	Tritt niemals auf

Diese Methode trennt die Verbindung der Klasse zur Anlage. Existiert noch keine Verbindung oder ist für **ScalarVis** noch kein **Control** angegeben, so kehrt die Methode einfach mit **True** zurück.

4.1.2.7 Function DoRefresh() As Boolean

Parameter: *Optional ByVal bQuick As Boolean = False*

bQuick	Optional; wenn hier **True** angegeben wird, werden nur der Ist-Wert und der Status der Tags ausgelesen (deutlich schneller)

Rückgabe:

True	Refresh erfolgreich durchgeführt
False	Fehler aufgetreten, Event **Warning** beachten

Mögliche Fehler/Warnungen:

pmsErrorConnectionLost	Verbindung zur Anlage abgerissen
pmsErrorConnect	Fehler beim Verbinden
pmsErrorRefresh	Interner Fehler aufgetreten

Mögliche Events:

pmsRefreshTags	Neue Tagwerte wurden gelesen und können abgefragt werden
pmsRefreshAlarms	Es sind aktuelle Alarme aufgetreten und können abgerufen werden.
pmsRefreshAlarmUpdate	Die Alarmhistorie wurde verändert.

Diese Methode holt die Daten von der Anlage und speichert sie in der Liste der internen Tags. Dabei werden *alle* relevanten Daten der zu aktualisierenden Tags ermittelt, sofern **bQuick**=**False**.

Bei ScalarVis werden die Grenz- und Rezeptwerte dabei von den folgenden Offsets gelesen:

Wert	Tagnummern
Ist-Wert	1 … 1000
Minimalwert 1 (Warn-Min)	1001 … 2000
Maximalwert 1 (Warn-Max)	2001 … 3000
Minimalwert 2 (SOC-Min)	3001 … 4000
Maximalwert 2 (SOC-Max)	4001 … 5000
Rezept Minimalwert 1 (Warn-Min)	5001 … 6000
Rezept Maximalwert 1 (Warn-Max)	6001 … 7000
Rezept Minimalwert 2 (SOC-Min)	7001 … 8000
Rezept Maximalwert 2 (SOC-Max)	8001 … 9000
Sollwert	10001 … 11000
Rezept Sollwert	11001 … 12000

Tabelle 1: Offsets der Sonder-Tags

Diese Zuordnung dieser Tags ist spezifisch für die Implementierung bei KoSa mit ScalarVis. Daher müssen in diesem Fall auch die verwendeten Anlagenserver dementsprechend eingestellt werden. Weiterhin ergibt sich daraus, dass die maximale Nummer eines normalen Tags nicht größer als 1000 sein kann.

Wird Wonderware verwendet, so gelten diese Einschränkungen nicht. Da der zugrunde liegende Aufbau völlig anders ist, fallen die o.g. Einschränkungen ersatzlos weg.

Sofern zur Zeit des Aufrufs keine Verbindung zur Anlage besteht, versucht die Methode, die Verbindung (wieder) aufzubauen. Das kann – bei mehreren Verbindungssätzen – dazu führen, dass der Server gewechselt wird, wenn der aktuelle nicht erreicht werden kann. Dabei werden die entsprechenden Events gesendet (siehe dazu **`ReConnect()`**).

Direkt nach dem Aufruf dieser Methode kann über die Eigenschaft **`Tags`** die aktuelle Liste der Werte abgeholt werden, sicherer ist es jedoch, eine Behandlungsroutine für das Event **`Refresh()`** der Klasse einzufügen und die Werte dort zu holen, wenn der Parameter **`pmsRefreshTags`** gesendet wurde.

Sofern bei den aktuell gelesenen Tags Alarme aufgetreten sind, wird das Event **`Refresh()`** nochmals gesendet, diesmal jedoch mit dem Parameter **`pmsRefreshAlarms`**. Dadurch lässt sich die Verarbeitung der Werte und der Alarme logisch voneinander trennen.

Weiterhin wird das Event **`Refresh()`** mit dem Parameter **`pmsRefreshAlarmUpdate`** gesendet, wenn sich die Alarmhistorie geändert hat. Dies kann auch dann vorkommen, wenn kein aktueller Alarm anliegt, sondern bisher bestehende Alarme verschwunden sind.

Eine sinnvolle Anwendung der Methode **`DoRefresh()`** liegt innerhalb einer Timer-Behandlung. Die Laufzeit dieses Timers kann aus der Eigenschaft **`RefreshTime`** oder nach eigenem Gutdünken eingestellt werden.

Hinweis zum optionalen Parameter **`blnQuick`**: Er sollte, um ein vernünftiges Reaktionsvermögen der Applikation zu ermöglichen, nur dann **`False`** sein, wenn die zusätzlichen Werte tatsächlich benötigt werden (z.B. nach dem Senden eines Rezepts oder nach dem Umstellen des Aktualisierungsbereichs). Der Zeitverlust, der durch das Einlesen aller Werte auftritt, liegt etwa bei Faktor 8-10![7]

[7] Bei Wonderware liegt der Faktor etwa bei 1,5 bis 2, also deutlich geringer.

4.1.2.8 Function DoSend() As Boolean

Parameter: *Keine*

Rückgabe:

True	Mindestens ein Datum wurde erfolgreich gesendet.
False	Beim Senden ist ein Fehler aufgetreten.

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Es existiert keine Verbindung zur Anlage und es konnte auch keine hergestellt werden.

Mögliche Events:

Refresh	Parameter: **pmsRefreshSend**

Diese Methode ist dazu gedacht, um geänderte Soll-, Grenz-, und/oder Rezeptwerte an die Anlage zu senden.

Zurzeit (April 2004) wird das Senden der Daten jedoch noch über externe Hilfsmittel direkt vorgenommen. Daher ist diese Methode noch ohne Funktion.

4.1.2.9 Function EnableActualizing() As Boolean

Parameter: *ByVal lTagFirst As Long,*
Optional ByVal lTagLast As Long = -1

lTagFirst	Erstes zu aktualisierendes Tag
lTagLast	Letztes zu aktualisierendes Tag (Default = **-1**)

Rückgabe:

True	Aktualisierungsflags wurden erfolgreich gesetzt
False	Es ist noch keine Tagliste definiert oder **lTagFirst<1** oder **lTagLast>TagLast**

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Bereichsangaben sind falsch (**lTagLast<lTagFirst** oder **lTagLast>TagLast**)

Zu Beginn (nach **Init()**) sind *alle* Tags deaktiviert. Um über die Methode **DoRefresh()** tatsächlich Daten zu erhalten, müssen die einzelnen Tags zuerst „scharf geschaltet“ werden. Dazu dient diese Methode; mit ihr können sowohl einzelne Tags (wenn **lTagLast** nicht angegeben wird), als auch ganze Bereiche (**lTagLast>lTagFirst**) eingestellt werden.

Bei den derart aktivierten Tags wird auch automatisch die Alarmierung eingeschaltet (siehe dazu auch **EnableAlarms()** und **DisableAlarms()**).

4.1.2.10 Function EnableAlarms() As Boolean

Parameter: *ByVal lTagFirst As Long,*
Optional ByVal lTagLast As Long = -1

lTagFirst	Erstes zu ändernde Tag
lTagLast	Letztes zu ändernde Tag (Default = **lTagFirst**)

Rückgabe:

True	Alarmierungsflags wurden erfolgreich gesetzt
False	Es ist noch keine Tagliste definiert oder **lTagFirst<1** oder **lTagLast>TagLast**

Diese Methode aktiviert die Alarmierung für den angegebenen Bereich von Tags. Sie muss nur dann aufgerufen werden, wenn zuvor mit **DisableAlarms** die Alarmierung spezieller Tags ausgeschaltet wurde. Ansonsten wird die Alarmierung beim Aktualisieren der Tags (siehe dazu auch **EnableActualizing()**) sofort mit aktiviert.

Der Rückgabewert ist **False**, wenn die Klasse noch nicht initialisiert wurde oder die Parameter ungültig sind. In diesem Fall werden keine Einstellungen geändert.

4.1.2.11 Function GetConnectionString() As String

Parameter: *Optional ByVal lIdx As Long = -1*

lIdx	Index des gesuchten Verbindungssatzes. Default ist der aktuell verwendete Satz.

Rückgabe:

ConnectionString	Der für den gesuchten Index generierte Verbindungsstring

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Der angegebene Index ist ungleich **-1** und kleiner als **1** oder größer als **ConnectionSetCount**

Mit dieser Methode kann jeder der durch **ConnectionString** generierten und intern gehaltenen Verbindungssätze als einzelner, normierter String gelesen werden.

Wurde **ConnectionString** noch nicht aufgerufen, oder die Daten gelöscht, so wird ein Leerstring zurückgeliefert.

4.1.2.12 Function GetName() As String

Parameter: *ByVal lTagNo As Long*

lTagNo	Nummer des Tags, dessen Name ermittelt werden soll

Rückgabe:

Tag-Name	Der Name wird bei der Abfrage direkt von der Anlage geholt.

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Die Tag-Nummer ist kleiner als 1 oder es besteht keine Verbindung zur Anlage

Mit dieser Methode kann von einem beliebigen Tag, also *auch von solchen, die nicht intern verwaltet werden* (siehe dazu **EnableActualizing()**), der Name abgefragt werden. Dabei ist es außerdem auch gleichgültig, ob das Tag aktualisierend ist oder nicht; es wird *auf jeden Fall* der aktuelle Name aus der Anlage gelesen.

Im Fehlerfall wird als Ergebnis ein leerer String zurückgeliefert.

4.1.2.13 Function GetText() As String

Parameter: *ByVal lTagNo As Long*

lTagNo	Nummer des Tags, dessen Text ermittelt werden soll

Rückgabe:

Aktueller Text	Der Text wird bei der Abfrage direkt ermittelt, so dass der zurzeit tatsächlich anliegende Wert geliefert wird.

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Die Tag-Nummer ist ungültig oder es besteht keine Verbindung zur Anlage

Mit dieser Methode kann von einem beliebigen Tag, also *auch von solchen, die nicht intern verwaltet werden* (siehe dazu **EnableActualizing()**), der Text abgefragt werden. Dabei ist es außerdem auch gleichgültig, ob das Tag aktualisierend ist oder nicht; es wird *auf jeden Fall* der aktuelle Text aus der Anlage gelesen.

Im Fehlerfall wird als Ergebnis ein leerer String zurückgeliefert.

4.1.2.14 Function GetValue() As Double

Parameter: *ByVal lTagNo As Long*

lTagNo	Nummer des Tags, dessen Ist-Wert ermittelt werden soll

Rückgabe:

Aktueller Ist-Wert	Der Wert wird bei der Abfrage direkt ermittelt, so dass der zurzeit tatsächlich anliegende Wert geliefert wird.

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Die Tag-Nummer ist ungültig oder es besteht keine Verbindung zur Anlage

Mit dieser Methode kann von einem beliebigen Tag, also auch von solchen, die nicht intern verwaltet werden (siehe **EnableActualizing()**), der aktuelle Ist-Wert abgefragt werden. Dabei ist es auch gleichgültig, ob das Tag aktualisierend ist, oder nicht; es wird auf jeden Fall der aktuelle Wert aus der Anlage gelesen. Auch wenn der Wert seine Grenzwerte (sofern ihm solche zugeordnet sind) überschreitet, werden keine Alarme generiert!

Im Fehlerfall wird als Ergebnis der Wert **-1E+38** zurückgeliefert.

4.1.2.15 Function Init() As Boolean

Parameter: *Keine*

Rückgabe:

True	Verbindung konnte hergestellt werden
False	Verbindung konnte nicht hergestellt werden, Event **Warning** bzw. Fehlermeldung beachten

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Es ist entweder noch kein **Control** angegeben oder noch kein **ConnectionString** definiert.
pmsErrorCantConnect	Es konnte keine Verbindung hergestellt werden

Mögliche Events:

Connection	Verbindung wurde hergestellt, Hostname wird als Parameter geliefert.

Diese Methode initialisiert die Klasse. Dabei wird die Verbindung zur Messdatenerfassung aufgenommen und die Abfrage der Tags vorbereitet.

Wenn die Verbindung hergestellt wurde, feuert die Methode das Event **Connection** mit dem Hostnamen als Parameter. Eine eventuell bereits bestehende Verbindung zur Anlage wird zuerst getrennt und danach die Verbindung wieder aufgebaut. Dabei kann auch der Server gewechselt werden, wenn der bisherige (bzw. der aktuelle in der Liste der Connection sets) nicht erreichbar ist. Daher sollte diese Methode nicht unnötig häufig aufgerufen werden, da das Anmelden – je nach Netzwerk und Beschäftigungsgrad des Servers – einige Sekunden in Anspruch nehmen kann.

Konnte keine Verbindung hergestellt werden (Kein Server erreichbar oder die Initialisierung der entsprechenden Komponente ist fehlgeschlagen), liefert die Funktion **`False`** und feuert ein Event **`Warning`** mit **`pmsErrorCantConnect`** für jeden nicht erreichbaren Server, jedoch nur dann, wenn mehrere Connection sets angegeben sind!

Ist dagegen nur ein Connection set vorhanden, so wird beim erfolglosen Verbinden ein Fehler mit dem Code **`pmsErrorCantConnect`** erzeugt.

4.1.2.16 Function PrepSendLimits() As Boolean

Parameter: *ByVal lTagNo As Long,*
ByVal dMin1 As Double,
ByVal dMin2 As Double,
ByVal dMax1 As Double,
ByVal dMax2 As Double,
Optional ByVal dSetPoint As Double = def_Invalid,
Optional ByVal lRecType As Long = -1

`lTagNo`	Nummer des Tags
`dMin1`	Untere Warngrenze
`dMin2`	Untere SOC-Grenze
`dMax1`	Obere Warngrenze
`dMax2`	Obere SOC-Grenze
`dSetPoint`	Sollwert (Default: keinen Sollwert einstellen)
`lRecType`	Recordtyp[8] (Default: kein Recordtyp)

Rückgabe:

`True`	erfolgreich
`False`	Fehler aufgetreten

Mögliche Fehler/Warnungen:

`pmsErrorInvalidParams`	Tagnummer ist außerhalb des mit **`EnableActualizing`** definierten Bereichs

Diese Methode stellt die Grenzwerte des durch **`lTagNo`** definierten Tags ein. Mit dieser Methode werden auf einmal alle Grenzwerte des Tags, sowie optional der Sollwert und der Recordtyp übergeben. Außerdem werden die Tag-internen Flags richtig eingestellt, sodass das Einstellen der **`Action`** nicht mehr notwendig ist.

Voraussetzung ist, dass das Tag „aktiv“ ist (durch **`EnableActualizing`**).

Zu diesem Zeitpunkt werden die Werte jedoch noch nicht zur Anlage gesendet. Dies geschieht erst bei der Ausführung von **`DoSend`**. Daher ist es sinnvoll, zuerst *alle* zu schreibenden Tags mit **`PrepSendLimits`** oder **`PrepSendRecipe`** einzustellen und danach **`DoSend`** auszuführen, um die Werte in die Anlage zu schreiben.

Sobald das Tag (mit dieser Methode oder direkt, siehe dazu Tag-Eigenschaft **`Action`**) den **`Action`**-Wert **`tacSendPLS`** enthält, werden sowohl die Rezept-, als auch die Grenzwerte beim Auslesen der Anlage ignoriert, um ein versehentliches Überschreiben der eingetragenen Werte (z.B. beim Aufruf durch ein Timer-Event) zu vermeiden.

[8] Der Recordtyp beschreibt die Art der Daten, die zur Prozessleitsteuerung gesendet werden. Diese Werte sind zurzeit (April 2004) noch nicht definiert.

4.1.2.17 Function PrepSendRecipe() As Boolean

Parameter: *ByVal lTagNo As Long,*
ByVal dMin1 As Double,
ByVal dMin2 As Double,
ByVal dMax1 As Double,
ByVal dMax2 As Double,
Optional ByVal dSetPoint As Double = def_Invalid,
Optional ByVal lRecType As Long = -1

lTagNo	Nummer des Tags
dMin1	Untere Rezept-Warngrenze
dMin2	Untere Rezept- SOC-Grenze
dMax1	Obere Rezept-Warngrenze
dMax2	Obere Rezept- SOC-Grenze
dSetPoint	Rezept-Sollwert (Default: keinen Sollwert einstellen)
lRecType	Recordtyp

Rückgabe:

True	erfolgreich
False	Fehler aufgetreten

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Tagnummer ist außerhalb des mit **EnableActualizing** definierten Bereichs

Diese Methode stellt die Rezeptwerte des durch **lTagNo** definierten Tags ein. Mit dieser Methode werden alle Rezeptwerte des Tags auf einmal übergeben. Außerdem werden die Tag-internen Flags richtig eingestellt, so dass das Einstellen der **Action** nicht mehr notwendig ist.

Voraussetzung ist, dass das Tag „aktiv“ ist (durch **EnableActualizing**).

Zu diesem Zeitpunkt werden die Werte jedoch noch nicht zur Anlage geschickt. Dies geschieht erst bei der Ausführung von **DoSend**. Daher ist es sinnvoll, zuerst alle zu schreibenden Tags mit **PrepSendLimits** und/oder **PrepSendRecipe** einzustellen und danach **DoSend** auszuführen, um die Werte in die Anlage zu schreiben.

Sobald das Tag (mit dieser Methode oder direkt, siehe dazu Tag-Eigenschaft **Action**) den **Action**-Wert **tacSendPLS** enthält, werden die Rezept- und Grenzwerte beim Auslesen der Anlage ignoriert, um ein versehentliches Überschreiben der Werte (z.B. beim Aufruf durch ein Timer-Event) zu vermeiden.

4.1.2.18 Function ReadArchiveValues() As Collection

Parameter: *ByVal lTagNum As Long,*
ByVal tFrom As Date,
ByVal tTo As Date,
Optional ByVal arcType As pmsArchiveTypes = pmsArchiveNormal

lTagNum	Referenznummer des gewünschten Tags
tFrom	Startzeit der Archivliste
tTo	Endzeit der Archivliste
arcType	**pmsArchiveNormal** für die normalen Archivwerte oder **pmsArchiveDetail** für die Störwerte (=alle Messungen)

Rückgabe:

Collection	Die gefundenen Archivwerte werden als Collection von Tags geliefert

Mögliche Fehler/Warnungen:

pmsErrorCantConnect	Keine Verbindung vorhanden
pmsErrorInvalidParams	Ungültiger **arcType** angegeben

Diese Methode liest Archivwerte eines Tags aus der Anlage. Dabei wird eine Collection von Tags erzeugt, die jeweils nur die Referenznummer (immer **lTagNum**), den jeweiligen Messwert, sowie die Uhrzeit dieser Messung enthalten.

Wird als **arcType pmsArchiveNormal** angegeben, so werden nur die zur Archivierung vorgesehenen Werte (bei ScalarVis im Abstand von ca. 180 Sekunden, das kann jedoch eingestellt werden) geliefert. Wird dagegen **pmsArchiveDetail** angegeben, so liefert die Methode die Werte aller Einzelmessungen. Allerdings werden diese Detailwerte nicht so lange aufbewahrt wie die Archivwerte.

4.1.2.19 Function ReadSOCMinMax() As Collection

Parameter: *ByVal lTagNum As Long,*
ByVal tFrom As Date,
ByVal tTo As Date

lTagNum	Referenznummer des gewünschten Tags
tFrom	Startzeit der Archivliste
tTo	Endzeit der Archivliste

Rückgabe:

Collection	Die gefundenen Werte werden als Collection vom Typ **Variant** geliefert. Bei Fehlern wird **Nothing** geliefert.

Mögliche Fehler/Warnungen:

pmsErrorCantConnect	Fehler, wenn keine Verbindung vorhanden ist

Diese Methode liefert den Maximalwert von SOC-Minimum und den Minimalwert von SOC-Maximum für ein Tag und eine bestimmte Zeitspanne. Dadurch lässt sich das kleinste Fenster des Sollbereichs ermitteln.

Als Ergebnis wird eine Collection geliefert, die zwei **Variants** vom internen Typ **Double** enthält. Der erste Wert (Index 1 der Collection) enthält das Maximum von SOC-Minimum (also den obersten Wert der Untergrenze), der zweite Wert (Index 2 der Collection) das Minimum von SOC-Maximum (den untersten Wert der Obergrenze).

Wurde eine ungültige Tagnummer (kleiner als 1 oder größer als **TagRange**) angegeben, so wird als Ergebnis anstelle der Collection **Nothing** geliefert.

Konnte einer der Werte nicht gelesen werden (weil z.B. die Anlage nicht entsprechend konfiguriert wurde), so enthält dieser Wert **-1E+38**.

4.1.2.20 Function ReConnect() As Boolean

Parameter: *Keine*

Rückgabe:

True	Eine Verbindung zur Anlage wurde hergestellt
False	Es konnte keine Verbindung hergestellt werden

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Es ist entweder noch kein **Control** angegeben oder noch kein **ConnectionString** definiert.
pmsErrorCantConnect	Fehler beim Verbinden

Mögliche Events:

Connection	Eine neue Verbindung wurde hergestellt; der Name des Hosts wird als Parameter geliefert.

Mit dieser Methode wird eine Verbindung zur Anlage hergestellt.

Kann die aktuelle Verbindung nicht aufgebaut werden *und* existieren mehrere Verbindungssätze (siehe dazu **ConnectionString**), so versucht die Methode nacheinander alle Verbindungssätze durch, bis entweder eine Verbindung hergestellt wurde oder alle Sätze gescheitert sind. Erst nachdem *alle* Verbindungssätze gescheitert sind, kehrt die Methode mit **False** zurück. Wurde eine Verbindung hergestellt, feuert die Methode das Event **Connection** mit dem Namen des verbundenen Servers als Parameter.

Sind mehrere Connection sets definiert, so generiert die Methode bei jedem gescheiterten Versuch die Warnung **pmsErrorConnect** mit dem Text „no connection to <Hostname>“. Daher sollte nicht vorschnell auf diese Meldung reagiert werden, sondern vielmehr das Ergebnis der Methode (**True** oder **False**) ausgewertet werden.

Ist dagegen nur ein Connection set angegeben, so wird beim Misslingen der Verbindung der Fehler **pmsErrorConnect** mit dem Text „no connection to <Hostname>“ generiert.

4.1.2.21 Function StartInternalTimer() As Boolean

Parameter: *Optional ByVal lInterval As Long = 0*

lInterval	Dauer des Timer-Intervalls in Sekunden (optional, Default: 0, wodurch die Zeit aus der Datenerfassung ermittelt wird)

Rückgabe:

True	Der Timer wurde gestartet, bzw. neu gestartet
False	Tritt niemals auf, bei Fehlern wird ein Error erzeugt

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Die aus der aktuellen Datenerfassung ermittelte Zeit ist kleiner als 1 Sekunde und damit ungültig.

Mögliche Events:

keine

Mit dieser Methode wird der interne Timer des Moduls gestartet. Dadurch wird im Abstand des Intervalls die Methode **DoRefresh** aufgerufen.

Wird kein Parameter oder **0** angegeben, so liest die Methode die **RefreshTime** des Datenerfassungsmoduls und verwendet diesen Wert.

4.1.2.22 Function StatisticGenerate() As Boolean

Parameter: *ByVal tFrom As Date,*
ByVal tTo As Date

tFrom	Anfangsdatum/Zeit für die Statistikdaten
tTo	Enddatum/Zeit für die Statistikdaten

Rückgabe:

True	Statistik-Werte wurden generiert
False	Es wurde kein Tag für die Statistik vorbereitet.

Mögliche Events:

Refresh	Parameter: **pmsRefreshStatistics**

Ermittelt statistische Werte für die angegebene Zeitdauer. Die ermittelten Statistikwerte sind **Durchschnitt**, **Minimum**, **Maximum**, **Anzahl**, **Startzeitpunkt** und **Endezeitpunkt** und werden direkt in die jeweiligen Tags eingetragen.

Damit die Werte ermittelt werden, muss vor dem Aufruf dieser Methode das **Action**-Flag der gewünschten Tags auf **tacGetStatistic** gestellt werden. Beispiel: Bei allen Tags die Statistikwerte der letzten 24 Stunden ermitteln:

```
For Each tag in PMS.Tags
   tag.Action = tag.Action + tacGetStatistic
Next tag
If PMS.StatisticGenerate(Now – 1, Now) Then [9]
   … Werte verwenden …
End If
```

Nach dem Ausführen der Methode enthalten die Felder **StatisticAverage**, **StatisticMinimum**, **StatisticMaximum**, **StatisticCount**, **StatisticDateStart** und **StatisticDateEnd** der Tags die ermittelten Werte.

Diese Werte werden auch durch ein erneutes **DoRefresh** nicht wieder gelöscht. Erst durch ein erneutes **StatisticGenerate** werden sie durch die neuen Werte überschrieben. Ein explizites Löschen der Werte kann nur durch die **clsTag**-Methoden **Clear** oder **ClearStat** in jedem einzelnen Tag erfolgen.

4.1.2.23 Function StopInternalTimer() As Boolean

Parameter: *Keine*

Rückgabe:

True	Der Timer wurde angehalten
False	Tritt niemals auf.

Mit dieser Methode wird der interne Timer des Moduls gestoppt. Dadurch wird das automatische Generieren von **Refresh**-Events beendet.

[9] **Now – 1** entspricht 24 Stunden

4.1.2.24 Function ValidData() As Boolean

Parameter: *Keine*

Rückgabe:

`True`	Alle notwendigen Parameter sind ausgefüllt
`False`	Die Verbindungsparameter wurden nicht oder ungültig definiert.

Diese Methode prüft, ob mindestens ein Satz Verbindungsparameter (siehe **`ConnectionString`**) definiert ist. Es wird *nicht* geprüft, ob mit diesen Daten eine tatsächliche Verbindung hergestellt werden kann!

Die Methode ist hauptsächlich zur internen Absicherung gegen Fehlbedienung gedacht; sie kann jedoch genau so gut von außen zur Prüfung verwendet werden.

4.1.3 Prozeduren

Im Folgenden werden die öffentlichen Prozeduren der Schnittstelle beschrieben. Auch diese sind als **`Public`** deklariert.

4.1.3.1 Sub WriteAlarmHistory()

Parameter: *Keine*

Diese Methode sichert die aktuelle Alarmhistorie auf dem Massenspeicher des Servers, auf dem die Datenerfassung läuft, in der Datei „**`AlarmHistory.txt`**“.

> Diese Methode kann zwar von der Anwendung aus aufgerufen werden und wird auch ausgeführt, sofern ScalarVis verwendet wird, jedoch werden die Alarme aktuell mit **`PMS_DbAlarm.exe`** in der PMS-Datenbank gespeichert und auch dort erwartet und gelesen.

4.1.4 Eigenschaften

Die im Folgenden beschriebenen Eigenschaften sind selbstverständlich ebenfalls alle als **`Public`** deklariert.

4.1.4.1 AlarmActual

Datentyp:

`Collection`	Liste der aktuellen Alarme

Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert eine Collection mit den beim letzten **`DoRefresh`** aufgetretenen Alarmen. Dabei besteht die Collection aus Elementen des Typs **`clsAlarm`**. Traten beim letzten **`DoRefresh`** keine Alarme auf, so ist die Collection leer.

4.1.4.2 AlarmCount

Datentyp:
Long Bereich: **0** … **TagCount**
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert die Anzahl der aktuellen (beim letzten **DoRefresh** aufgetretenen) Alarme.

4.1.4.3 AlarmHist

Datentyp:
Collection Liste der kompletten Alarmhistorie.
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert eine Collection mit den Alarmen der kompletten Alarmhistorie. Dabei besteht die Collection genau wie bei **AlarmActual** aus Elementen des Typs **clsAlarm**.

Ein Alarm wird maximal zweimal in die Liste eingetragen: Einmal, wenn er das erste Mal auftritt; das zweite Mal, wenn er wieder verschwindet.

Die Historie wird bei Verwendung von ScalarVis auf dem Rechner gespeichert, auf dem das Datenerfassungsmodul läuft (siehe dazu **WriteAlarmHistory**). Daraus ergibt sich zum einen, dass die Historie unabhängig von der Laufzeit des Clients ist; zum zweiten, dass die Historie beim Wechsel des Servers auch gewechselt wird.

Da diese Lösung recht proprietär ist und ausschließlich bei ScalarVis verwendet werden kann, wird sie heute nicht mehr verwendet. Vielmehr läuft heute pro Anlage (mindestens) einmal das Programm „**PMS_DbAlarm.exe**“, das die Alarme über die aktuellen Rezepte ermittelt und in der Datenbank ablegt. Die Client-Programme holen sich dann auch die Alarme aus der Datenbank.

In der nächsten Version werden daher die Alarm-bezogenen Methoden, Funktionen und Eigenschaften nicht mehr aktiv sein.

4.1.4.4 AlarmHistCount

Datentyp:
Long Bereich: **0** … **AlarmHistSize**
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert die Anzahl der Alarme in der kompletten Alarmhistorie. Die Anzahl kann maximal so groß wie **AlarmHistSize** werden.

4.1.4.5 AlarmHistIndex

Datentyp:
Long Bereich: **0 .. 2.147.483.647**
Zugriff:
lesend und schreibend
Mögliche Fehler/Warnungen:
keine

Liefert / setzt den aktuellen Index der Alarmhistorie. Beim nächsten **AlarmHistUpdate** werden die Alarme ab diesem Index geliefert.

4.1.4.6 AlarmHistSize

Datentyp:
Long Bereich: **0 .. 2.147.483.647**
Zugriff:
lesend und schreibend
Mögliche Fehler/Warnungen:
pmsErrorInvalidParams Wert ist kleiner als 0

Die Größe der Alarmhistorie (die maximale Anzahl an Alarmen, die gespeichert werden, bevor die ältesten gelöscht werden).

Der Standardwert ist 1000, d.h. sobald der 1001. Alarm erscheint, wird der erste gelöscht, usw.

Die Größe lässt sich zwischen 0 und 2.147.483.647 einstellen, der Default von 1000 ist jedoch ein guter Kompromiss zwischen Speicherverbrauch und sinnvoller Historie.

Wird beim Einstellen ein Wert kleiner als 0 angegeben, wird der Fehler **pmsErrorInvalidParams** erzeugt und der bisherige Wert wird nicht verändert.

4.1.4.7 AlarmHistUpdate

Datentyp:
Collection
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert eine Collection mit den Alarmen der Alarmhistorie, wobei nur die seit dem letzten Aufruf von **AlarmHist** oder **AlarmHistUpdate** neu aufgetretenen Alarme geliefert werden. Ansonsten wird die gleiche Collection wie bei **AlarmHist** geliefert.

4.1.4.8 CollectorType

Datentyp:

pmsCollectorType

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidCollector	Ungültiger Typ angegeben (nur beim Zuweisen)

Liefert den aktuellen Datenerfassungstyp, bzw. stellt ihn ein. Die möglichen Typen sind unter dem Typ **pmsCollectorType** beschrieben.

Beim Zuweisen eines gültigen Wertes, der vom bisher eingestellten abweicht, wird auch automatisch der interne Timer gestoppt, sofern er zuvor aktiv war. Allerdings wird der Timer nicht neu gestartet; diese Maßnahme bleibt der Anwendung vorbehalten, um eine sichere Synchronisierung zu gewährleisten.

4.1.4.9 ConnectionSetCount

Datentyp:

Long	Bereich: **0 … n**

Zugriff:

nur lesend

Mögliche Fehler/Warnungen:

keine

Liefert die Anzahl der durch **ConnectionString** festgelegten Verbindungssätze. Wurde noch kein **ConnectionString** angegeben, so ist die Anzahl 0.

4.1.4.10 ConnectionSetIndex

Datentyp:

Long	Bereich: **0**, **1 … ConnectionSetCount**

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams	Neuer Index ist größer als **ConnectionSetCount** oder kleiner als 1.

ConnectionSetIndex ist der Index des aktiven Verbindungssatzes. Der erste Verbindungssatz hat den Index 1. Existiert zurzeit noch kein Verbindungssatz, wird als Index 0 geliefert.

Beim *Setzen* ist ein Wert kleiner als 1 nicht erlaubt, die Werte müssen immer zwischen 1 und **ConnectionSetCount** liegen.

4.1.4.11 ConnectionString

Datentyp:

String

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams Es wurde kein Host angegeben

Diese Eigenschaft definiert die Verbindungsparameter zur Anlage. Der String, der zugewiesen wird, muss gewissen Regeln entsprechen: Er besteht aus ein bis sechs Teilen, die den Typ, Host, Port, Benutzernamen, Kennwort und den verwendeten Betrieb festlegen. Untereinander sind diese Elemente durch Semikolons (;) voneinander getrennt. Die einzelnen Elemente sind:

Element[10]	Beispiel	Defaultwert	Beschreibung
TYP	**TYP=W** **TYP=M** **TYP=P**		Angabe der Verbindung, die verwendet werden soll. **M** steht dabei für MessDas (oder ScalarVis), **W** für Wonderware und **P** für PI[11].
HOST **HST** **HOSTNAME**	**HOST=XYZ** **HOST=ABC,XYZ**		Der Name des Rechners, mit dem sich die Klasse verbinden soll.
PORT **PRT** **PORTNR**	**PORT=1024** **PORT=1024,32768**	**32768**	Der Port, über den das Datenerfassungssystem angesprochen werden soll.[12]
USER **USR** **USERNAME**	**USER=Gast** **USER=Gast,Admin**	**nogast**	Der Benutzername, mit dem die Anmeldung an das Datenerfassungssystem erfolgen soll.
PWD **PASSWD** **PASSWORD**	**PWD=geheim** **PWD=kein,geheim**	**nogast**	Das Kennwort zum Benutzernamen.
BTR **BETR** **BETRIEB**	**BTR=P2K** **BETR=P2D** **BETRIEB=P3K**		Der Betrieb, mit dem sich das Modul verbinden soll. Dieser Wert wird nur beim Typ **W** (Wonderware) benötigt und ausgewertet.

Tabelle 2: Elemente für ConnectionString

Alle anderen Angaben im String werden ignoriert.

Das Element „**TYP**“ wird nicht an die Unterelemente weitergereicht. Vielmehr dient es dazu, in **PMSKopplung** das zu verwendende Verbindungsmodul zu definieren. Aus diesem Grund wird es auch beim Lesen des **ConnectionStrings** nicht zurückgeliefert. Weiterhin ist es hierbei nicht sinnvoll, mehrere Werte anzugeben, da nur der erste ausgewertet wird. Der Typ wird in der Registrierung gespeichert.

Das Element „**BETRIEB**“ wird zurzeit nur bei Wonderware ausgewertet. Es stellt dadurch den Betrieb ein, mit dem das System arbeiten soll. Dies ist wichtig, da das Modul „Wonderware“ einige Einstellungen aus der PMS-Datenbank holt und diese anhand des Betriebes unterschieden werden müssen. Auch hierbei wird nur der erste Parameter verwendet und weitere ignoriert.

ScalarVis ignoriert dieses Element.

Die Reihenfolge der Elemente ist beliebig; sie werden innerhalb des Moduls selbständig sortiert.

[10] Sind in der Tabelle mehrere Elemente angegeben, so werden alle akzeptiert. Die Groß- und Kleinschreibung ist nur beim Kennwort relevant.

[11] PI ist und wird wahrscheinlich auch niemals implementiert. Da zum Zeitpunkt des grundlegenden Designs dies aber noch nicht bekannt war, existiert eine Implementation für PI, die jedoch nur leere Daten liefert.

[12] Der Port ist bei **Wonderware** überflüssig. Da er bei **ScalarVis** jedoch benötigt wird, wird sich diese Einstellung in der Klasse **PMSKopplung** nicht ändern.

Ein typischer **ConnectionString** sieht daher folgendermaßen aus:

Beispiel 1:

```
PMS.ConnectionString = "Typ=M;Host=Srv0815;Port=32768;User=Hans;Pwd=Gretel"
```

Um mehrere Verbindungssätze zu definieren, können in jedem Element weitere Werte — durch Komma getrennt — angegeben werden.

Beispiel 2:

```
"Host=Srv1,Srv2,Srv3;Port=32768,1024;User=Hans;Pwd=Gretel"
```

Dieses Beispiel definiert drei Verbindungssätze:

Nr.	HOST	PORT	USER	PWD
1.	Srv1	32768	Hans	Gretel
2.	Srv2	1024	Hans	Gretel
3.	Srv3	1024	Hans	Gretel

Die Anzahl der Verbindungssätze ergibt sich aus dem Element mit der höchsten Anzahl von Werten; im Beispiel das Element **„Host“** mit 3 Werten: **„Srv1“**, **„Srv2“**, **„Srv3“**. Enthält ein Element weniger Werte, so wird das jeweils letzte kopiert. In der Tabelle sind die invertiert dargestellten Werte derartige Kopien.

Eine weitere Möglichkeit, mehrere Verbindungssätze zu erstellen, ist, *vor das erste Zeichen* im String ein „+“ (Pluszeichen) einzufügen. Dadurch werden die bisherigen Verbindungssätze nicht gelöscht und der oder die neuen nur zugefügt.

Zu beachten ist hierbei, dass – wenn nicht alle Parameter angegeben wurden – als Defaults nicht die zuvor übergebenen Werte, sondern die Standardwerte verwendet werden.

Beispiel 3:

```
PMS.ConnectionString = "Host=Srv15,Srv16;Port=32768;User=Hans;Pwd=Gretel"
PMS.ConnectionString = "+Host=Srv17;Port=1234"
```

Nach diesen beiden Kommandos existieren drei Verbindungssätze:

Nr.	HOST	PORT	USER	PWD
1.	Srv15	32768	Hans	Gretel
2.	Srv16	32768	Hans	Gretel
3.	Srv17	1234	nogast	nogast

Die invertiert dargestellten Felder sind wieder Kopien aus dem ersten Set. Die grau hinterlegten sind dagegen Standardwerte.

Die letzte Möglichkeit der Manipulation ist es, als **ConnectionString** nur ein **„-“** (Minuszeichen) ohne weiteren Text anzugeben. Dadurch werden alle vorhandenen Verbindungssätze gelöscht. Beispiel:

```
PMS.ConnectionString = "-"
```

Die bisherigen Angaben bezogen sich alle auf das Setzen der Eigenschaft. Auch wenn es nicht besonders relevant ist, kann die Eigenschaft jedoch auch abgefragt werden. Dabei wird der *zur Zeit aktuelle* Verbindungssatz geliefert.

Beispiel:

```
PMS.ConnectionString = "Typ=M;Port=32768,1024,9999;Host=Srv1,Srv2,Srv3"
PMS.Init
Debug.Print "Verbindungssatz: " & PMS.ConnectionString
```

Angenommen, dass der erste Server nicht, der zweite aber erreichbar ist, wird folgender Text angezeigt:

```
Verbindungssatz: Host=Srv2;Port=1024;User=nogast;Pwd=nogast
```

Hierbei wird auch gezeigt, dass die Reihenfolge der Elemente unabhängig von der Zuweisung ist.

4.1.4.12 EnabledTags

Datentyp:

Collection — Liste der zu aktualisierenden Tags als Collection

Zugriff:

nur lesend

Liefert eine Collection, die nur die Tags enthält, die aktiv sind, also mit **EnableActualizing** aktiviert wurden. Der Vorteil gegenüber der Eigenschaft **Tags** ist der, dass hiermit kleinere Listen transportiert werden, der Nachteil, dass der Zusammenhang zwischen dem Index der Collection und der Tagnummer verloren geht (Allerdings ist die Tagnummer innerhalb des Tags selbstverständlich noch vorhanden).

4.1.4.13 Hostname

Datentyp:

String — Name des aktuellen Servers

Zugriff:

nur lesend

Liefert den Namen des aktuell aktiven Hosts. Ist noch kein Connection set definiert, so wird ein leerer String geliefert.

4.1.4.14 Password

Datentyp:

String — Das aktuelle Kennwort im Klartext

Zugriff:

nur lesend

Liefert das aktuell verwendete Kennwort. Ist noch kein Connection set definiert, so wird ein leerer String geliefert.

4.1.4.15 Port

Datentyp:

Variant — Ergebnis wird als **Variant** mit dem internen Typ **Long** geliefert

Zugriff:

nur lesend

Liefert die aktuell eingestellte Portnummer. Ist noch kein Connection set definiert, so wird **-1** geliefert.

4.1.4.16 RefreshTime

Datentyp:

Long — Die eingestellte Refresh-Zeit in Sekunden

Zugriff:

nur lesend

Liefert die Zeit in Sekunden, nach der die Werte aktualisierbar sind, bzw. aktualisiert werden. Es ist zwar möglich, eine höhere Abfragefrequenz festzulegen, jedoch nicht sinnvoll, da sich die Daten nur im Takt von **RefreshTime** Sekunden ändern können. Dazwischen werden jedes Mal dieselben Werte geliefert. Existiert noch keine Verbindung zur Anlage, so wird **0** geliefert.

4.1.4.17 TagBegin

Datentyp:

Long — Index des ersten auszuwertenden Tags

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams — Der übergebene Wert ist kleiner als **1** oder größer als **TagLast**

Liefert die kleinste Tagnummer, deren Daten aktualisiert werden. Wurde noch kein Bereich (mit der Methode **EnableActualizing**, siehe Seite 12) definiert, so wird 0 geliefert.

Dieser Wert wird innerhalb der Klasse nicht (mehr) verwendet.

4.1.4.18 TagCount

Datentyp:

Long — Anzahl der auszuwertenden Tags

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams — Der Wert plus **TagBegin** ist größer als **TagLast**

Liefert die Anzahl der abzufragenden Tags (**TagEnd – TagBegin + 1**).

Durch Einstellen dieses Wertes wird indirekt auch **TagEnd** beeinflusst.

Dieser Wert wird innerhalb der Klasse nicht (mehr) verwendet.

4.1.4.19 TagEnd

Datentyp:

Long — Index des letzten auszuwertenden Tags

Zugriff:

lesend und schreibend

Mögliche Fehler/Warnungen:

pmsErrorInvalidParams — Der übergebene Wert ist kleiner als **TagBegin** oder größer als **TagLast**

Liefert die größte eingestellte Tag-Referenznummer.

Dieser Wert wird innerhalb der Klasse nicht (mehr) verwendet.

4.1.4.20 TagLast

Datentyp:
Long — Der höchstmögliche Tag-Wert (in ScalarVis fest als 1000 definiert)

Zugriff:
nur lesend

Mögliche Fehler/Warnungen:
keine

Liefert die maximal mögliche Tagnummer (In ScalarVis zurzeit fest definiert als 1000)

4.1.4.21 TagRecordset

Datentyp:
ADODB.Recordset — Liste der aktuellen Tag-Werte als **ADODB.Recordset**

Zugriff:
nur lesend

Mögliche Fehler/Warnungen:
keine

Diese Eigenschaft liefert die Liste der zuletzt aktualisierten Tags in Form eines ADODB Recordsets. Dadurch lassen sich die Werte leicht in so genannte Data bound Controls wie True DB Grid, o. ä. einfügen.

Für jedes Element der Klasse **clsTag** wird ein Feld mit dem Datentyp VARCHAR(255) generiert. Dies sind:

Feldname	Beschreibung
Ref	Die Tagnummer im MES.
Name	Der Name der Messstelle.
FilteredName	Ebenfalls der Name der Messstelle, jedoch ohne führende Betriebskennung und ohne anhängende Typenkennung.
ID	Der erste Teil des Namens (ID)
Description	Der zweite Teil des Namens (Bezeichnung)
Unit	Die SI-Einheit, sofern vorhanden
Func	Tag-interne Funktionsnummer (nur bei ScalarVis interessant)
DBA	Der Beginn des Darstellungsbereichs
DBE	Das Ende des Darstellungsbereichs
Value	Der Istwert
TagText	Der evtl. zugewiesene Tagtext.
SetPoint	Der Sollwert
LimitMin2	Untere Grenze (SOC)
LimitMin1	Untere Warngrenze
LimitMax1	Obere Warngrenze
LimitMax2	Obere Grenze (SOC)
RecipeSetPoint	wie zuvor, jedoch die Werte, die im Rezept stehen.
RecipeLimitMin2	
RecipeLimitMin1	
RecipeLimitMax1	
RecipeLimitMax2	
State	Status der Messstelle

Action	Gewünschte Aktion
RecordType	Art des Records beim Senden an PLS
StatisticAverage	Statistik: Mittelwert
StatisticMinimum	Statistik: Minimalwert
StatisticMaximum	Statistik: Maximalwert
StatisticCount	Statistik: Anzahl der ausgewerteten Messstellen
StatisticDateStart	Statistik: Anfang des Zeitraums
StatisticDateEnd	Statistik: Ende des Zeitraums
LastUpdate	Zeitpunkt der letzten Aktualisierung

Tabelle 3: Recordset-Felder

Für jeden Tag der Collection **Tags** wird dann ein eigener Record in das Recordset eingefügt.

4.1.4.22 Tags

Datentyp:
Collection — Liste der aktuellen Tag-Werte als Collection
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert die interne Liste von Tags. Ist diese leer, bzw. noch nicht definiert, so wird eine leere Collection geliefert.

Die Collection enthält, sofern die Klasse mit **Init** initialisiert wurde, immer **TagLast** Elemente. Siehe dazu auch **EnabledTags**.

4.1.4.23 TimerRunning

Datentyp:
Boolean — **True**, wenn der interne Timer läuft, sonst **False**
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert **True**, wenn der interne Timer durch **StartInternalTimer** gestartet wurde, ansonsten **False**.

4.1.4.24 Username

Datentyp:
String — Der aktuell aktive Benutzername
Zugriff:
nur lesend
Mögliche Fehler/Warnungen:
keine

Liefert den aktuell verwendeten Benutzernamen. Ist noch kein Connection set definiert, so wird ein leerer String geliefert.

4.1.5 Ereignisse

Die Klasse **PMSKopplung**, bzw. die internen Unterklassen generieren bei verschiedenen Gelegenheiten Ereignisse (so genannte Events), die vom aufrufenden Programm abgefangen und behandelt werden können.

4.1.5.1 Event Connection(sHost As String)

Dieses Ereignis wird von der Methode **ReConnect** (und daher auch indirekt von **Connect** und **Init** sowie allen anderen Methoden, die eine neue Verbindung initiieren) gefeuert, sobald eine funktionierende Verbindung aufgebaut wurde.

Als Parameter wird in **sHost** der Name des Hosts geliefert.

4.1.5.2 Event Refresh(RefreshType As pmsRefreshType)

Dieses Ereignis wird von den Methoden **DoRefresh**, **StatisticGenerate** und **DoSend** gefeuert, wenn die Daten verändert wurden und im Client aktualisiert werden müssen. **RefreshType** kann die folgenden Werte enthalten:

Wert	Beschreibung
pmsRefreshTags	Die normalen Tag-Daten wurden aktualisiert und können jetzt ausgelesen werden.
pmsRefreshStatistics	Die Statistik-Daten wurden generiert und können jetzt ausgelesen werden.
pmsRefreshAlarms	Die Alarmhistorie wurde aktualisiert und kann jetzt ausgelesen werden.
pmsRefreshSend	Die Daten wurden an die Anlage gesendet.

Tabelle 4: Refreshtypen

4.1.5.3 Event Warning(Warningcode As pmsErrorcode, sInfo As String)

Dieses Ereignis wird dann gefeuert, wenn dem Hauptprogramm eine Warnung mitgeteilt werden muss. Eine Warnung bedeutet, dass ein unerwartetes Ereignis aufgetreten ist, das jedoch nicht unbedingt einen Abbruch des Programms erzwingen muss. Folgende Warnungen können auftreten:

Warningcode	Textinfo	Bedeutung
pmsErrorCantConnect[13]	**no connection to <Host>**	Es konnte keine Verbindung zu **Host** aufgebaut werden, evtl. jedoch zu einem anderen Server.
pmsErrorConnectionLost	**connection lost**	Die bisherige Verbindung besteht nicht mehr.

Tabelle 5: Warnungen

[13] Dieser Code tritt auch als Fehler auf

4.1.5.4 »Normale« Fehler

Zusätzlich zu den mit **Warning** gesendeten Warnungen werden auch noch übliche VB-Fehler generiert. Außer den vom System generierten definiert **PMS_Mod** noch die folgenden eigenen Fehlercodes:

Fehlercode	Beschreibungstext	Bedeutung
pmsErrorInvalidParams (= 513)	verschiedene	Mindestens einer der übergebenen Parameter ist ungültig, der Beschreibungstext enthält eine genauere Spezifikation des Fehlers.
pmsErrorCantConnect (= 514)	**no connection to Host** oder **can't connect**	Es konnte keine Verbindung zu **Host** aufgebaut werden
pmsErrorInvalidCollector (= 516)	**invalid collector #NNN**	Es wurde ein falscher Datenerfassungstyp angegeben; gültig sind nur **pmsCollMessdas**, **pmsCollPI** und **pmsCollWonderware**, in **NNN** wird der numerische Wert des angegebenen Typs mitgeteilt.

Tabelle 6: Fehlercodes

Zu den in Klammern angegebenen Fehlercodes wird immer noch die VB-Konstante **vbObjectError** (entspricht **&H80040000**, dezimal **-2147221504**) addiert. Diese Konstante muss daher beim Auswerten beachtet werden.

4.1.6 Enumerationen

Die folgenden Enumerationen[14] werden in **PMS_Mod** definiert und verwendet.

4.1.6.1 pmsArchiveTypes

Typ zur Auswahl der gewünschten Archivwerte bei **ReadArchiveValues**.

Wert	Beschreibung
pmsArchiveNormal	normale Archivwerte; die Werte, die in die Archivierung eingetragen werden. Diese Werte haben einen höheren zeitlichen Abstand. Dafür sind jedoch weitaus ältere Werte möglich als bei „Detail“.
pmsArchiveDetail	Stör- oder Detailwerte; die tatsächlichen Messwerte, die von ScalarVis ermittelt wurden. Hier werden alle Messwerte geliefert, jedoch nur bis zu einem geringeren Maximalalter.

Tabelle 7: Archivierungstypen

Dieser Datentyp ist nur bei ScalarVis relevant, Wonderware liefert auch bei älteren Daten die hohe Auflösung.

4.1.6.2 pmsCollectorType

Dieser Datentyp unterscheidet die verschiedenen Datenerfassungsmodule.

Wert	Beschreibung
pmsCollInvalid	ungültige Angabe; wird nur während der Initialisierung verwendet.
pmsCollMessdas	Das aktive Datenerfassungsmodul ist **ScalarVis** (**MessDas**)
pmsCollWonderware	Das aktive Datenerfassungsmodul ist **Wonderware**
pmsCollPI	Das aktive Datenerfassungsmodul ist **PI**

Tabelle 8: Datenerfassungsmodultypen

4.1.6.3 pmsErrorcode

Dieser Datentyp wird beim Event **Warning**, bzw. als Fehlercode bei **Err.Raise** mitgeschickt.

Wert	Beschreibung
pmsErrorInvalidParams	Ein oder mehrere Parameter sind ungültig (VB-Fehler)
pmsErrorCantConnect	Aufnahme der Verbindung zu ScalarVis ist fehlgeschlagen (Warnung oder VB-Fehler)
pmsErrorConnectionLost	Während des Ausführens von **DoRefresh** wurde festgestellt, dass die Verbindung zu ScalarVis nicht mehr besteht. Je nach Einstellung wird ein automatischer **ReConnect** versucht, wobei auch der Server gewechselt werden kann.
pmsErrorInvalidCollector	Der Wert des Collectors ist ungültig (siehe **pmsCollectorType**)

Tabelle 9: Fehler- und Warnungscodes

[14] also durch Symbole dargestellte Zahlenwerte

4.1.6.4 pmsRefreshType

Dieser Datentyp wird beim Event **`Refresh`** übermittelt und definiert somit, welche Art von Refresh tatsächlich ausgeführt wurde.

Wert	Beschreibung
`pmsRefreshTags`	Normale Tag-Daten wurden aktualisiert.
`pmsRefreshStatistics`	Statistik-Daten wurden generiert.
`pmsRefreshAlarms`	Die Alarmhistorie wurde aktualisiert.
`pmsRefreshSend`	Daten wurden an die Anlage gesendet.

Tabelle 10: Refreshtypen

4.2 clsTag

Die Klasse clsTag ist der eigentliche Container für die Daten, die von der Anlage geliefert werden. Sie enthält (private) Felder für alle relevanten Elemente, auf die mittels Eigenschaften zugegriffen werden kann.

4.2.1 Prozeduren

4.2.1.1 Sub Clear()

Parameter: *Optional ByVal bClearAll As Boolean = True*

Parameter:

`bClearAll` **`True`**: auch **`Ref`**, **`Action`** und Statistikwerte löschen
`False`: **`Ref`**, **`Action`** und Statistikwerte bleiben erhalten

Diese Methode ermöglicht es, den Tag zu löschen. Wird **`bClearAll`** mit **`False`** angegeben, so bleiben die Felder **`Ref`**, **`Action`**, sowie alle statistischen Werte erhalten.

4.2.1.2 Sub ClearStat()

Parameter: *Keine*

Hiermit werden ausschließlich die *Statistik*werte des Tags gelöscht. Dazu gehören die folgenden Elemente:

Element	zugewiesener Wert
`StatisticAverage`	**`-1E+38`**
`StatisticMinimum`	**`-1E+38`**
`StatisticMaximum`	**`-1E+38`**
`StatisticCount`	**`0`**
`StatisticDateStart`	**`0`**
`StatisticDateEnd`	**`0`**
`Action`	Flag **`tacGetStatistic`** wird gelöscht

Tabelle 11: gelöschte Statistikwerte

4.2.2 Eigenschaften

4.2.2.1 Action

Datentyp:

Long

Zugriff:

lesend und schreibend

Enthält als Bitmuster die verschiedenen möglichen Aktionen (siehe dazu auch den Datentyp **tacAction**):

Bit	Beschreibung
tacGetStatistic	markiert dieses Tag zum Lesen der Statistikwerte (siehe **StatisticGenerate** in **PMSKopplung**).
tacReadCurveSet	markiert dieses Tag zum Lesen von Kurven[15].
tacSendPLS	markiert dieses Tag zum Senden an die Anlage (siehe **DoSend**).
tacSendSecRequest	noch nicht implementiert; geplant zur Bestätigung bei Sendevorgängen.

Tabelle 12: Aktions-Flags

Eine Zuweisung von anderen als den hier genannten Flags ist unter Umständen möglich, muss dann aber von der Applikation verwaltet werden. Prinzipiell sollte es vermieden werden, andere als die hier gelisteten Bits zu bearbeiten.

4.2.2.2 ActualizeEnabled

Datentyp:

Boolean

Zugriff:

lesend und schreibend

Dieser Schalter aktiviert und deaktiviert das entsprechende Flag. Nur wenn es aktiviert ist (**True**), wird es beim nächsten Auslesen der Daten in **DoRefresh** gefüllt.

4.2.2.3 AlarmEnabled

Datentyp:

Boolean

Zugriff:

lesend und schreibend

Dieser Schalter aktiviert die Alarmierung für dieses Tag. Nur wenn sowohl dieser Schalter, als auch **ActualizeEnabled** auf **True** stehen, kann von diesem Tag ein Alarm generiert werden.

[15] Zurzeit (April 2004) noch nicht implementiert

4.2.2.4 AllItems

Datentyp:

Collection

Zugriff:

nur lesend

Liefert eine Collection mit allen Daten dieses Tags. Die Collection enthält die folgenden Elemente:

Index	Key	Index	Key
1	Ref	16	recipeSetPoint
2	Name	17	recipeLimitMin2
3	ID	18	recipeLimitMin1
4	Description	19	recipeLimitMax1
5	Unit	20	recipeLimitMax2
6	Func	21	State
7	DBA	22	Action
8	DBE	23	RecordType
9	Value	24	StatisticAverage
10	TagText	25	StatisticMinimum
11	SetPoint	26	StatisticMaximum
12	LimitMin2	27	StatisticCount
13	LimitMin1	28	StatisticDateStart
14	LimitMax1	29	StatisticDateEnd
15	LimitMax2	30	LastUpdate

Tabelle 13: Datenelemente clsTag

Die einzelnen Elemente der Collection können sowohl über den Index, als auch über die in der Tabelle angegebenen Keys angesprochen werden.

4.2.2.5 AllNames

Datentyp:

Collection

Zugriff:

nur lesend

Liefert eine Collection mit den Namen aller Datenfelder dieses Tags als Strings (siehe dazu auch **AllItems**). Auch hier sind die Elemente sowohl über den Index, als auch über die Keys ansprechbar.

Anmerkung: Bei dieser Collection sind die Inhalte mit den Keys identisch.

4.2.2.6 AllTypes

Datentyp:

Collection

Zugriff:

nur lesend

Liefert eine Collection mit den Datentypen aller Datenfelder dieses Tags als Strings. Diese Collection enthält die folgenden Elemente:

Index	Datentyp	Index	Datentyp
1	Long	16	Double
2	String	17	Double
3	String	18	Double
4	String	19	Double
5	String	20	Double
6	Long	21	Long
7	Double	22	Long
8	Double	23	Long
9	Double	24	Double
10	String	25	Double
11	Double	26	Double
12	Double	27	Long
13	Double	28	Date
14	Double	29	Date
15	Double	30	Date

Tabelle 14: Datentypen clsTag

Auch hier sind die Elemente sowohl über den Index, als auch über die (in Tabelle 13 definierten) Keys ansprechbar.

4.2.2.7 DBA

Datentyp:

Double

Zugriff:

lesend und schreibend

Beginn des Darstellungsbereichs (z.B. für Kurven). Dieser Wert wird beim Refresh aus der Anlage gelesen. Auch wenn der Wert innerhalb des Tags geändert werden kann, wird der Wert z. Zt. *nicht* an die Anlage gesendet.

4.2.2.8 DBE

Datentyp:

Double

Zugriff:

lesend und schreibend

Ende des Darstellungsbereichs (z.B. für Kurven). Dieser Wert wird beim Refresh aus der Anlage gelesen. Auch wenn der Wert innerhalb des Tags geändert werden kann, wird der Wert z. Zt. *nicht* an die Anlage gesendet.

4.2.2.9 Description

Datentyp:

`String`

Zugriff:

lesend und schreibend

Beim Auslesen der Daten aus der Anlage wird bei der Zuweisung des Namens (siehe dazu **`Name`**) dieser in zwei Teile gesplittet:

1. die **`ID`** und
2. einen zusätzlichen Beschreibungstext (**`Description`**).

Die Teilung wird folgendermaßen ausgeführt: Die ersten 10 Zeichen von **`Name`** werden in **`ID`** kopiert, ab Zeichen 11 dagegen in **`Description`**. Danach werden bei beiden Zeichenketten die eventuell verbleibenden Leerzeichen am Anfang und Ende entfernt.

4.2.2.10 Func

Datentyp:

`Long`

Zugriff:

lesend und schreibend

Diese Eigenschaft ist spezifisch für ScalarVis. Bei ScalarVis muss jedem Tag eine bestimmte Funktion zugewiesen werden. Über diese Funktion wird definiert, wie die Ausgangsdaten aus den Rohdaten des Tags generiert werden. Zu weiterführenden Informationen zu diesen Funktionen sei hier auf das Handbuch „ScalarVis - Administration“, Abschnitt „Funktions-Bibliothek“, verwiesen.

Auch wenn diese Eigenschaft geändert werden kann, so hat diese Änderung zurzeit noch keinerlei Auswirkungen auf den darunter liegenden Service.

4.2.2.11 HasAlarm

Datentyp:

`Boolean`

Zugriff:

nur lesend

Nachdem beim Auslesen die Daten in das Tag übertragen wurden, kann mit dieser Eigenschaft kontrolliert werden, ob in diesem Tag ein Alarm vorliegt. Dies ist jedoch nur dann möglich, wenn die Eigenschaft **`AlarmEnabled`** **`True`** ist. Ansonsten wird — und natürlich auch, wenn kein Alarm aufgetreten ist — **`False`** geliefert.

4.2.2.12 ID

Datentyp:

`String`

Zugriff:

lesend und schreibend

Die Eigenschaft ID ist ein normaler String, der jedoch beim Zuweisen des Namens (siehe Eigenschaft **`Name`**) eingestellt wird. Für detailliertere Informationen siehe unter **`Description`**.

4.2.2.13 LastUpdate

Datentyp:

Date

Zugriff:

lesend und schreibend

Datum und Uhrzeit der letzten Aktualisierung im Format **Date**. Der Wert wird bei jeder Aktualisierung eingestellt.

4.2.2.14 LimitMax1

Datentyp:

Double

Zugriff:

lesend und schreibend

Die obere Warngrenze für dieses Tag.

4.2.2.15 LimitMax2

Datentyp:

Double

Zugriff:

lesend und schreibend

Die obere SOC-Grenze für dieses Tag.

4.2.2.16 LimitMin1

Datentyp:

Double

Zugriff:

lesend und schreibend

Die untere Warngrenze für dieses Tag.

4.2.2.17 LimitMin2

Datentyp:

Double

Zugriff:

lesend und schreibend

Die untere SOC-Grenze für dieses Tag.

4.2.2.18 Name

Datentyp:

String

Zugriff:

lesend und schreibend

Der Name des Tags. Bei der Zuweisung des Namens werden auch **ID** und **Description** eingestellt.

4.2.2.19 recipeLimitMax1

Datentyp:

Double

Zugriff:

lesend und schreibend

Die obere Warngrenze des Rezepts für dieses Tag.

4.2.2.20 recipeLimitMax2

Datentyp:

Double

Zugriff:

lesend und schreibend

Die obere SOC-Grenze des Rezepts für dieses Tag.

4.2.2.21 recipeLimitMin1

Datentyp:

Double

Zugriff:

lesend und schreibend

Die untere Warngrenze des Rezepts für dieses Tag.

4.2.2.22 recipeLimitMin2

Datentyp:

Double

Zugriff:

lesend und schreibend

Die untere SOC-Grenze des Rezepts für dieses Tag.

4.2.2.23 recipeSetPoint

Datentyp:

Double

Zugriff:

lesend und schreibend

Der Sollwert aus dem Rezept für dieses Tag.

4.2.2.24 RecordType

Datentyp:

Long

Zugriff:

lesend und schreibend

Zur Datenübermittlung an das PLS muss der **RecordType** angegeben werden, da hierdurch in der PLS entsprechende Werte eingestellt werden. Zurzeit wird diese Eigenschaft zwar von der Anlage ins Tag, jedoch noch nicht in der Gegenrichtung transportiert. Dies wird beim Senden der Daten (siehe dazu auch **PMSKopplung** / **DoSend**) jedoch noch implementiert.

4.2.2.25 Ref

Datentyp:

Long

Zugriff:

lesend und schreibend

Diese Eigenschaft enthält die laufende Nummer des Tags. Sie ist identisch mit der Nummer, unter der die Daten in der Anlage referenziert sind.

4.2.2.26 SetPoint

Datentyp:

Double

Zugriff:

lesend und schreibend

Der Sollwert für dieses Tag.

4.2.2.27 State

Datentyp:

Long

Zugriff:

lesend und schreibend

Enthält den numerischen Status dieses Tags. Dieser Status umfasst die verschiedenen möglichen Fehlercodes. Sowohl beim Zuweisen, als auch bei der Abfrage wird der Wert mit **StateMask** gefiltert, so dass nur die dort definierten Bits übernommen, bzw. geliefert werden. Die möglichen Bits sind unter dem Enum **tagState** beschrieben.

4.2.2.28 StateMask

Datentyp:

Long

Zugriff:

nur lesend

Liefert eine Bitmaske, in der alle Bits aus dem Enum **tagState** enthalten sind.

4.2.2.29 StatisticAverage

Datentyp:

Double

Zugriff:

lesend und schreibend

Enthält den statistischen Durchschnittswert dieses Tags. Obwohl der Wert frei zugewiesen werden kann, wird er am sinnvollsten durch die **PMSKopplung**-Methode **StatisticGenerate** belegt.

4.2.2.30 StatisticCount

Datentyp:

Long

Zugriff:

lesend und schreibend

Enthält die Anzahl der bei der letzten statistischen Auswertung verwendeten Werte (siehe dazu auch **StatisticGenerate**). Die Eigenschaft kann jedoch auch jederzeit beliebig frei belegt werden.

4.2.2.31 StatisticDateEnd

Datentyp:

Date

Zugriff:

lesend und schreibend

Enthält den Zeitpunkt der letzten Messung in der statistischen Auswertung (siehe dazu auch **StatisticGenerate**). Die Eigenschaft kann jedoch auch jederzeit beliebig frei belegt werden.

4.2.2.32 StatisticDateStart

Datentyp:

Date

Zugriff:

lesend und schreibend

Enthält den Zeitpunkt der ersten Messung in der statistischen Auswertung (siehe dazu auch **StatisticGenerate**). Die Eigenschaft kann jedoch auch jederzeit beliebig frei belegt werden.

4.2.2.33 StatisticMaximum

Datentyp:

Double

Zugriff:

lesend und schreibend

Enthält den Maximalwert aus der zuletzt erfassten Statistik (siehe auch **StatisticGenerate**). Die Eigenschaft kann jedoch auch jederzeit beliebig frei belegt werden.

4.2.2.34 StatisticMinimum

Datentyp:

Double

Zugriff:

lesend und schreibend

Enthält den Minimalwert aus der zuletzt erfassten Statistik (siehe auch **StatisticGenerate**). Die Eigenschaft kann jedoch auch jederzeit beliebig frei belegt werden.

4.2.2.35 TagText

Datentyp:

String

Zugriff:

lesend und schreibend

Enthält den Text des Tags. Beim Senden von Rezepten wird in diesem String zurzeit der **RecordType** in folgendem Format abgelegt:

```
RCT0000
```

wobei **RCT** das Kürzel für ***R**e**C**ord**T**ype* ist und die Ziffern durch den Wert des RecordType's ersetzt werden. Ansonsten kann dieser String für beliebige Zwecke verwendet werden.

4.2.2.36 Unit

Datentyp:

String

Zugriff:

lesend und schreibend

Enthält die im System angegebene SI-Einheit des Messwertes. Üblicherweise wird hier ein führendes Leerzeichen angegeben, damit die Einheit direkt an den Wert angehängt werden kann. Dies ist jedoch nicht zwingend vorgeschrieben, so dass es sicherer ist, die Formatierung manuell vorzunehmen:

```
strWert = tag.Value & " " & LTrim(tag.Unit)
```

4.2.2.37 Value

Datentyp:

Double

Zugriff:

lesend und schreibend

Dieses Element enthält das wichtigste Element der Klasse: den eigentlichen aktuellen Messwert. Dieser Wert wird beim Aktualisieren über **PMSKopplung** / **DoRefresh** immer aktualisiert, auch wenn der Quickmodus eingestellt ist.

4.2.3 Enums

4.2.3.1 tagState

Dieser Datentyp wird in der Eigenschaft **State** verwendet. Er stellt den entsprechenden Status des Tags anhand der folgenden Bits dar.

Name	Bit	Beschreibung
tstOkay	**keins**	Alles in Ordnung, kein Fehlerzustand
tstSystemAlarm	**0**	Systemalarm / Leitungsbruch, o.ä.
tstLimitMin2	**1**	SOC-Minimum verletzt (kritische Unterschreitung des Sollbereichs)
tstLimitMin1	**2**	Warn-Minimum verletzt (unkritische Unterschreitung des Sollbereichs)
tstLimitMax1	**3**	Warn-Maximum verletzt (unkritische Überschreitung des Sollbereichs)
tstLimitMax2	**4**	SOC-Maximum verletzt (kritische Überschreitung des Sollbereichs)
tstConnectionError	**7**	Fehler in der Kopplung
tstAlarmGone	**12**	Alarm wieder verschwunden (erscheint nur in **clsAlarm** und dort auch nur in der Historie)

Tabelle 15: Datentyp tagState

4.2.3.2 tagAction

Dieser Datentyp stellt die verschiedenen Modi dar, die in der Eigenschaft **Action** eingestellt werden können.

Name	Wert	Beschreibung
tacNone	**&H0000**	Keine Aktion
tacGetStatistic	**&H0100**	Statistikwerte ermitteln
tacReadCurveSet	**&H0200**	Kurvensatz lesen (noch nicht implementiert)
tacSendPLS	**&H0400**	Daten (Rezept- und/oder Grenzwerte) ans PLS senden
tacSendSecRequest	**&H0800**	Sicherheitsabfrage für **tacSendPLS** (nicht implementiert)
tacNoAlarm	**&H4000**	Keine Aktion, sondern ein Flag, dass dieses Tag keine Alarme erzeugt.
tacNoActualize	**&H8000**	Keine Aktion, sondern ein Flag, dass die Aktualisierung dieses Tags unterdrückt wird.

Tabelle 16: Datentyp tagAction

Die beiden Flags **tacNoAlarm** und **tacNoActualize** können über die Eigenschaft **Action** nicht gesetzt oder gelöscht werden. Sie werden intern für den Status „Aktualisieren“ sowie „Alarmieren“ benötigt.

4.3 clsAlarm

Die Klasse **clsAlarm** dient zum Transport von Alarmen, die von der Anlage geliefert werden. Sie enthält private Felder für alle relevanten Elemente, auf die wie üblich mittels öffentlicher Eigenschaften zugegriffen werden kann.

Die Klasse wird nicht mehr benötigt, da die Alarme über die Datenbank abgehandelt werden. Trotzdem werden sie in **PMSKopplung** zurzeit noch generiert. Eventuell wird diese Funktionalität später einmal entfernt, um die Performance zu steigern.

4.3.1 Methoden

4.3.1.1 Public Sub Copy()

Parameter: *ByRef src As clsAlarm,*
Optional ByVal bGone As Boolean = True

src	Quellobjekt, dessen Werte kopiert werden sollen
bGone	**True**: das kopierte Element wird als *Beenden des Alarms* angelegt (Default) **False**: das Element wird so kopiert, wie es ist

Diese Methode kopiert die Werte aus dem Quellobjekt **src**, welches ebenfalls vom Typ **clsAlarm** ist, in das aktuelle Objekt. Die Methode wird intern hauptsächlich zum Aufbau der Historie verwendet, kann jedoch auch von außen benutzt werden.

Wenn **bGone** gesetzt ist (Defaultwert), wird im Status (siehe dazu **State**) das Bit **tstAlarmGone** gesetzt, wodurch dieses Alarmobjekt als „Beenden des Alarms“ gekennzeichnet wird.

Ansonsten werden einfach alle Elemente des Quellobjekts **src** in das aktuelle Objekt kopiert.

4.3.1.2 Public Function Comp() As Boolean

Parameter: *ByRef src As clsAlarm*

src	Quellobjekt, mit dem das aktuelle Objekt verglichen wird.

Rückgabe:

True	Die Alarme sind gleich
False	Die Alarme sind ungleich

Diese Methode vergleicht die Werte aus dem Quellobjekt **src** mit den Werten aus dem aktuellen Objekt, um festzustellen, ob die beiden Alarme gleich sind.

„Gleich“ bedeutet an dieser Stelle, dass sowohl die Tagnummer, als auch der Status übereinstimmen. Beim Vergleich des Status wird das Bit **tstAlarmGone** ignoriert, so dass kommende und gehende Alarme als identisch erkannt werden können.

4.3.1.3 Public Function Fill() As Boolean

Parameter: *ByVal tag As clsTag,*
Optional ByVal nRef As Long = -1

tag	Das Tag, aus dem ein Alarm generiert werden soll.
nRef	Die Referenznummer für den zu erstellenden Alarm

Rückgabe:

True	Es wurde ein Alarm aus dem Tag generiert.
False	Es wurde kein Alarm generiert.

Diese Methode generiert ein Alarmobjekt, wenn das Tag einen Status ungleich 0 enthält. Dabei werden die relevanten Daten (**TagNo**, **Text** und **State**) aus dem Tag ins Alarmobjekt kopiert und außerdem die aktuelle Uhrzeit des Alarms eingestellt.

Wenn in **nRef** keine Referenznummer, bzw. ein Wert kleiner als **0** angegeben wird, generiert die Methode einen Zufallswert im Bereich **0…32768**. Dieses Vorgehen sollte jedoch vermieden werden, da so keine Garantie für eine eindeutige Nummerierung besteht.

Wenn der Status des Tags 0 (=**tstOkay**) ist, wird kein Alarm generiert und die Methode kehrt mit **False** zurück.

Beispiel:

```
1: Dim tag As clsTag, al As clsAlarm, col As Collection
2: Set al = New clsAlarm
3: Set col = New Collection
4: PMS.DoRefresh
5: For Each tag In PMS.Tags
6:    If al.Fill(tag, col.Count + 1) Then
7:       col.Add al
8:    End If
9: Next tag
```

In diesem Beispiel wird zuerst die Tagliste aktualisiert (Zeile 4) und danach jedes Tag (Zeile 5) auf seinen Status überprüft (Zeile 6). Enthält das Tag einen Alarm, dann wurde in **al.Fill()** das Alarmobjekt gefüllt und wird in Zeile 7 in die Collection übernommen.

4.3.2 Eigenschaften

4.3.2.1 AllItems

Datentyp:

`Collection`

Zugriff:

nur lesend

Liefert eine Collection mit den acht Datenelementen dieses Alarms. Die Collection enthält die folgenden Elemente:

Index	Element	Index	Element
1	`Ref`	5	`TagNo`
2	`BackRef`	6	`State`
3	`Errorcode`	7	`StateText`
4	`Time`	8	`Text`

Tabelle 17: Datenelemente clsAlarm

4.3.2.2 AllNames

Datentyp:

`Collection`

Zugriff:

nur lesend

Liefert eine Collection mit den Namen aller Datenfelder dieses Tags als Strings (siehe dazu **`AllItems`**). Hierbei sind die Schlüssel und Werte der Collection identisch.

4.3.2.3 AllTypes

Datentyp:

`Collection`

Zugriff:

nur lesend

Liefert eine Collection mit den Datentypen aller Datenfelder dieses Alarms als Strings. Die Collection enthält die folgenden Elemente:

Index	Datentyp	Index	Datentyp
1	`Long`	5	`Long`
2	`Long`	6	`Long`
3	`Long`	7	`String`
4	`Date`	8	`String`

Tabelle 18: Datentypen clsAlarm

4.3.2.4 BackRef

Datentyp:

Long

Zugriff:

lesend und schreibend

Diese Eigenschaft ist nur dann sinnvoll, wenn dieses Alarmobjekt einen verschwindenden Alarm darstellt. In diesem Fall enthält **`BackRef`** die Referenznummer des entsprechenden Start-Alarms (siehe dazu **`Ref`** und **`Fill()`**).

Dieser Wert sollte nicht manuell verändert werden, da sonst die Zusammenhänge der Alarmhistorie gestört werden.

4.3.2.5 Errorcode

Datentyp:

Long

Zugriff:

lesend und schreibend

Diese Eigenschaft wird zurzeit noch nicht verwendet (Ausnahme: Beim Initialisieren der Klasse und in der Methode **`Fill()`** wird sie mit 0 belegt). Sie steht daher zu freien Verfügung.

4.3.2.6 Gone

Datentyp:

`Boolean`

Zugriff:

nur lesend

Diese Eigenschaft liefert **`True`**, wenn das Alarmobjekt nicht das Auftreten, sondern das *Verschwinden* eines Alarms beschreibt. Das bedeutet, dass in **`State`** das Bit **`tstAlarmGone`** gesetzt ist.

Diese Eigenschaft kann nur innerhalb der Historie **`True`** enthalten, da in der normalen Alarmliste ein verschwundener Alarm einfach nicht mehr eingetragen wird.

4.3.2.7 Ref

Datentyp:

Long

Zugriff:

lesend und schreibend

Diese Eigenschaft enthält die Referenznummer dieses Alarms. Da bei VB6 das Klassenkonzept noch nicht in allen Formen ausgereift ist, muss von außen dafür gesorgt werden, dass die Nummer eindeutig ist (siehe dazu das Beispiel zu **`Fill()`**, **„`col.Count + 1`“**).

4.3.2.8 State

Datentyp:

Long

Zugriff:

lesend und schreibend

Hier wird bei **Fill()** der Status des Tags eingetragen. Bei dieser Zuweisung wird auch sofort der **StateText** generiert.

4.3.2.9 StateText

Datentyp:

String

Zugriff:

lesend und schreibend

Beim Zuweisen eines Status über die Eigenschaft **State** wird hier abhängig von **State** einer der folgenden Texte eingetragen:

Status	Text
tstOkay	
tstSystemAlarm	# SYSTEM #
tstLimitMin2	LL
tstLimitMin1	L
tstLimitMax1	H
tstLimitMax2	HH
tstConnectionError	# CONNECTION #

Tabelle 19: Statustexte in clsAlarm

Dieser Text kann zwar auch unabhängig vom Status verändert werden, jedoch sollte darauf verzichtet werden, um eine konsistente Anzeige des Status zu gewährleisten.

4.3.2.10 Time

Datentyp:

Date

Zugriff:

lesend und schreibend

Hier wird bei **Fill()** Datum und Uhrzeit des Alarms eingetragen.

4.3.2.11 TagNo

Datentyp:

Long

Zugriff:

lesend und schreibend

Hier wird bei **Fill()** die Nummer des Tags (Eigenschaft **Ref** der Klasse **clsTag**) eingetragen. Dadurch wird ein eindeutiger Zusammenhang zwischen Tag und Alarm ermöglicht.

4.3.2.12 Text

Datentyp:

`String`

Zugriff:

lesend und schreibend

Hier wird bei **`Fill()`** der Name des Tags eingetragen. Die Eigenschaft hat rein informativen Charakter.

4.3.2.13 TextString

Datentyp:

`String`

Zugriff:

lesend und schreibend

Diese Eigenschaft dient hauptsächlich zum Importieren und Exportieren der Daten des Alarms. Dazu werden beim Auslesen von **`TextString`** alle Elemente des Alarms in einen String formatiert, jeweils getrennt durch „¶“ (‚Pilcrow’ oder Absatzzeichen, Chr(182)). Die Reihenfolge der Elemente ist:

Nr.	Feld	Beschreibung
1	**`Ref`**	Referenznummer des Alarms
2	**`Errorcode`**	Fehlercode
3	**`Time`**	Datum & Uhrzeit in länderunabhängigem Format
4	**`BackRef`**	Referenznummer des erscheinenden Alarms beim Verschwinden
5	**`TagNo`**	Referenznummer des Tags, bei dem der Alarm auftrat
6	**`State`**	Status des zugehörigen Tags
7	**`StateText`**	Status des zugehörigen Tags in Textform
8	**`Text`**	Name des zugehörigen Tags

Tabelle 20: TextString-Felder

Die Nummern der Elemente beziehen sich auf das folgende Schema:

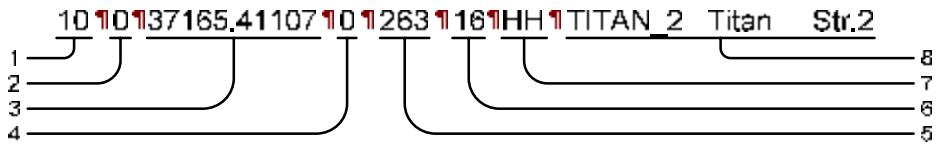

Abbildung 3: TextString-Format

Dieser Beispieltext stellt einen gültigen Alarm dar. Es lässt sich durch sein einfaches Textformat leicht in eine Datei schreiben, um z.B. eine statische Alarmhistorie aufzubauen.

Entsprechendes gilt selbstverständlich bei der Zuweisung eines Strings an **`TextString`**. Die Eigenschaft erwartet alle o.g. Elemente in der angegebenen Reihenfolge. Es ist leider nahezu unmöglich, hierbei eine umfassende Validierung aller Parameter zu implementieren, da VB in diesem Bereich sehr „gutmütig“ ist und viele – eigentlich falsche – Parameter stillschweigend konvertiert. Daher wird nur kontrolliert, ob der übergebene String wirklich 8 Parameter enthält und der Text, der den Zeitpunkt beschreibt, als Datum interpretiert werden kann.

Anmerkung: Da die Gutmütigkeit von VB nicht soweit reicht, dass US-Versionen deutsches Datum lesen können oder umgekehrt, musste das Datum in einer länderunabhängigen Form gespeichert werden. Dazu wird das im Wert enthaltene Datum als Ganzzahl, gefolgt von einem Dezimalpunkt und der Uhrzeit formatiert.

Hauptsächlich wird diese Methode von **PMSKopplung** intern zum Speichern und Laden der Alarmhistorie verwendet.

Da die Alarmhistorie inzwischen über das Modul „**PMS_DbAlarm**“ in der Datenbank verwaltet wird, wird **TextString** zwar noch unterstützt, jedoch nicht mehr benötigt.

4.4 clsWWare

Die interne Klasse **clsWWare** behandelt die Datenübertragung vom und zum System „Wonderware“. Die folgende Beschreibung behandelt weniger die Bedienung (da diese Klasse sowieso nicht von außen direkt ansprechbar ist), sondern vielmehr die internen Abläufe, sowie die notwendigen SQL-Statements.

4.4.1 Interne Variablen

Alle Zustände und Daten der Klasse werden in internen Variablen gehalten.

Variablenname	Typ	Beschreibung
m_Conn	ADODB.Connection	Verbindung zum SQL-Server mit den Datenbanken „Runtime“ und „Holding“
m_rsTagRel	ADODB.Recordset	Enthält die PMS-Tabelle, mit der die Verbindung zwischen Tagnummer und Tagname hergestellt werden.
m_bConnectState	Boolean	Der aktuelle Status der Verbindung zur Datenbank.
m_bLockChanges	Boolean	Schalter, der den automatischen Serverwechsel unterdrückt.
m_cConnectionParams	Collection	Sammlung aller Parameter zur Verbindung.
m_lNumConnectionSets	Long	Anzahl der Verbindungs-Sets.
m_lCurrConnection	Long	Index des aktuellen Verbindungs-Sets.
m_cTags	Collection	Liste der Tags, die durch Wonderware gefüllt werden.
m_lTagBegin	Long	Startindex der Tagnummern, die aktualisiert werden.
m_lTagEnd	Long	Endindex der Tagnummern, die aktualisiert werden.
m_cAlarms	Collection	Liste der aktuellen Alarme.
m_sBetrieb	String	Bezeichnung des aktuellen Betriebs.
m_cAlarmHist	Collection	Liste, die die Alarmhistorie enthält.
m_lAlarmHistRefSize	Long	Maximale Größe der Alarmhistorie.
m_lAlarmHistLastRead	Long	Die Position der Alarmhistorie, von der zuletzt gelesen wurde.

Tabelle 21: clsWWare, interne Variablen

4.4.2 Interne Prozeduren

4.4.2.1 Private Sub Class_Initialize()

Parameter: *Keine*

Wird bei der Erstellung der Klasse (z.B. durch **New**) aufgerufen. Hier werden nur die internen Variablen in einen definierten Zustand gebracht.

4.4.2.2 Private Sub Class_Terminate()

Parameter: *Keine*

Wird beim Zerstören der Klasse (z.B. durch **Set xxx = Nothing**) aufgerufen. Schließt sowohl das Recordset **m_rsTagRel**, als auch die Connection **m_Conn**. Weiterhin werden die verschiedenen Collections gelöscht.

4.4.3 Interne Funktionen

4.4.3.1 Private Function LoadTagRelations() As ADODB.Recordset

Parameter: *Optional ByVal sBetrieb As String = ""*

Diese Funktion ist die einzige Stelle in **clsWWare**, in der auf die PMS-Datenbank zugegriffen wird. Hier wird das Recordset **m_rsTagRel** gefüllt, das für die Konvertierung der Tagnummern zu den in Wonderware verwendeten Tagnamen, sowie für den Zusammenhang zwischen Basis- und Sollwert-Tag verwendet wird.

Je nachdem, ob bereits ein Betrieb festgelegt wurde oder nicht, werden entweder nur die Zeilen gelesen, die zum Betrieb gehören, oder aber alle.

Das zugehörige SQL-Statement:

```
SELECT [ID] AS [TagNo], [Value] AS [TagName], [Setpoint]
 FROM [wwTags]
( WHERE [Betrieb] = '<Betrieb>' )
 ORDER BY [ID]
```

Wenn in **m_sBetrieb** noch kein Betrieb angegeben ist, wird ohne die **WHERE**-Klausel gearbeitet, d.h., es werden alle bekannten Tags geliefert. Ansonsten wird das Recordset nur mit den Tags des aktuellen Betriebes gefüllt.

Diese Funktion wird auch dann aufgerufen, wenn der Betrieb gewechselt wird. Das bedeutet, wenn zu Beginn noch kein Betrieb definiert war und daher alle Tags geladen wurden, dass die Liste beim Zuweisen eines Betriebes automatisch berichtigt wird.

4.4.3.2 Private Function SetpointName() As String

Parameter: *ByVal lRef As Long*

Liefert den Namen des Tags, das den Sollwert für das aktuelle Tag liefert. Dies wird (mit der Tagnummer in **lRef**) über das Recordset **m_rsTagRel** ermittelt. Existiert für die angegebene Tagnummer kein Sollwert, so wird ein Leerstring geliefert.

4.4.3.3 Private Function StateConvert() As Long

Parameter: *ByVal lQuality As Long*
ByVal lQualityDetail As Long

Diese Funktion übersetzt den Status einer Messung mit dem Detailstatus 24 (bedeutet **„IOServer communication failed“**) in den Status **tstConnectionError** (**0x80**). Bei jedem anderen Status wird **tstOkay** (**0x00**) geliefert.

4.4.3.4 Private Function TagName() As String

Parameter: *ByVal lRef As Long*

Liefert den Namen des Tags mit der Tagnummer **lRef**. Wie schon bei **SetpointName()** wird dies über das Recordset **m_rsTagRel** ermittelt. Existiert die angegebene Tagnummer nicht, so wird ein Leerstring geliefert.

4.4.4 Öffentliche Prozeduren[16]

4.4.4.1 Public Sub AdjustAlarmHistory()

Parameter: *Keine*

Verwaltet den FIFO-Speicher der Alarmhistorie. Dabei wird kontrolliert, ob die definierte Größe der Historie überschritten wurde. Wenn ja, werden alte Einträge gelöscht und die Indices angepasst.

Anmerkung: Diese Prozedur dient der „alten“ Alarmverwaltung und wird nicht mehr benötigt.

4.4.4.2 Public Sub AddAlarmsToHistory()

Parameter: *ByVal cNewAlarms As Collection*

Fügt aus der Liste der aktuellen Alarme (**cNewAlarms**) diejenigen in die Historie ein, die sich — verglichen mit der letzten Liste — geändert haben. Dabei werden auch Alarme, die nun nicht mehr vorhanden sind, als **GONE** eingefügt.

Dadurch wird die Historie aktualisiert und aktuell gehalten.

Anmerkung: Diese Prozedur dient der „alten“ Alarmverwaltung und wird nicht mehr benötigt.

4.4.4.3 Public Sub WriteAlarmHistory()

Parameter: *Optional ByVal sFilename As String = ""*

Diese Prozedur wird nur wegen der notwendigen Schnittstelle zur Verfügung gestellt. Sie ist ohne Funktion.

[16] „Öffentlich“ bedeutet in diesem Zusammenhang nur, dass die Prozeduren, Funktionen, Eigenschaften, etc. in der kompletten DLL sichtbar sind, nicht, dass sie über das COM-Interface freigegeben sind!

4.4.5 Öffentliche Funktionen[17]

4.4.5.1 Public Function ChangeServer() As Boolean

Parameter: *Keine*

Diese Funktion wechselt den Server, sofern mehrere Verbindungssets vorhanden sind. Sie liefert **`True`**, wenn der Wechsel erfolgreich war, ansonsten **`False`**. Auch wenn nur ein Verbindungsset vorliegt, wird **`False`** geliefert.

Bei einem erfolgreichen Wechsel wird die Funktion **`Init()`** aufgerufen, die die grundlegende Initialisierung vornimmt.

Schlägt der Wechsel oder die anschließende Initialisierung fehl, so wird der vorherige Zustand wieder restauriert.

4.4.5.2 Public Function CheckConnection() As Boolean

Parameter: *Keine*

Prüft, ob eine Verbindung zur Wonderware-Datenbank besteht. Wenn ja, wird **`True`** zurückgegeben, sonst **`False`**.

Ist ein Verbindungsobjekt (**`m_Conn`**) vorhanden, aber nicht verbunden (**`.State <> adStateOpen`**), bedeutet das, dass die Verbindung unterbrochen wurde. In diesem Fall wird außer dem Ergebnis **`False`** auch noch ein Event **`Warning(pmsErrorConnectionLost, "connection lost")`** generiert. Die Funktion versucht *nicht*, die Verbindung wieder herzustellen!

4.4.5.3 Public Function Connect() As Boolean

Parameter: *Keine*

Im Gegensatz zu **`CheckConnection()`** stellt **`Connect()`** eine Verbindung her, sofern sie noch nicht besteht. Dazu wird die interne Variable **`m_bConnectState`** geprüft. Ist sie **`False`**, so wird mittels **`ReConnect()`** versucht, eine Verbindung herzustellen. Ist **`m_bConnectState`** dagegen bereits **`True`**, so liefert die Funktion einfach **`True`** zurück.

Daraus folgt, dass die Funktion ohne große zeitliche Nachteile aufgerufen werden kann, um eine Verbindung sicherzustellen. Besteht sie bereits, so ist **`Connect()`** sehr schnell fertig. Im anderen Fall muss sowieso eine Verbindung erstellt werden.

[17] „Öffentlich“ bedeutet in diesem Zusammenhang nur, dass die Prozeduren, Funktionen, Eigenschaften, etc. in der kompletten DLL sichtbar sind, nicht, dass sie über das COM-Interface freigegeben sind!

4.4.5.4 Public Function DisableActualizing() As Boolean

Parameter: *Optional ByVal lTagFirst As Long = -1*
Optional ByVal lTagLast As Long = -1

Diese Funktion deaktiviert die angegebenen Tags. Das bedeutet, dass sie ab dem nächsten **DoRefresh()** nicht mehr aktualisiert werden.

Wird **lTagFirst** nicht angegeben, so wird dafür die erste bekannte Tagnummer verwendet. Wird **lTagLast** nicht angegeben, wird die letzte bekannte Tagnummer verwendet.

Nun werden alle Tags, deren Nummern im so definierten Bereich liegen, deaktiviert. Dadurch werden die zuletzt ermittelten Werte jedoch nicht verändert, sondern belassen.

Wurde mindestens ein Tag deaktiviert, liefert die Funktion **True**, sonst **False**.

4.4.5.5 Public Function DisableAlarms() As Boolean

Parameter: *Optional ByVal lTagFirst As Long = -1*
Optional ByVal lTagLast As Long = -1

Adäquat zu **DisableActualizing()** wird hier nur die Alarmierung der Tags deaktiviert. Dadurch generieren diese Tags keine Alarme mehr.

Wurde bei mindestens einem Tag die Alarmierung deaktiviert, liefert die Funktion **True**, sonst **False**.

Da die gesamte Alarmierung insgesamt nicht mehr auf dieser Ebene, sondern über das Programm **„PMS_DbAlarm.exe“** realisiert wird, wird diese interne Alarmierung nicht mehr benötigt.

4.4.5.6 Public Function Disconnect() As Boolean

Parameter: *Keine*

Schließt und trennt die Verbindung zur Wonderware-DB und zerstört das Objekt **m_Conn**. Außerdem wird **m_bConnectState** auf **False** gesetzt.

Wird danach versucht, auf die Daten zuzugreifen, so wird die Klasse versuchen, sofort wieder eine Verbindung herzustellen!

4.4.5.7 Public Function DoRefresh() As Boolean

Parameter: *Optional ByVal bQuick As Boolean = False*

Dies ist eine der wichtigsten Funktionen der gesamten Klasse. Sie wird durch **DoRefresh()** der Klasse **PMSKopplung** aufgerufen. Hier werden alle Tags, die aktualisierend sind, mit Werten gefüllt.

Zuerst wird geprüft, ob eine Verbindung mit der Datenbank möglich ist. Wenn nicht, wird ein Fehler erzeugt und dadurch auch die Funktion abgebrochen.

Danach wird die interne Collection **m_cAlarms** initialisiert.

Als nächstes werden Strings mit Listen der Tagnamen und (wenn **bQuick=False**) der Setpointnamen so zusammengestellt, dass sie für SQL-Statements mit **„WHERE [TagName] IN (…)“** verwendet werden können.

Danach werden die entsprechenden Statements ausgeführt und in den Recordsets **rsTg** und (wenn **bQuick=False**) **rsSp** gespeichert.

Dann werden in einer Schleife über alle gefundene Tags die aktuellen Daten ermittelt. Dies sind auf jeden Fall der aktuelle Messwert, der Status und die aktuelle Uhrzeit.

Ist **`bQuick=False`**, werden auch noch Tagname, Bezeichnung, Maßeinheit, interne Funktion, sowie der Kurvenbereich (DBA & DBE) geholt.

Ist **`bQuick=False`** und das Tag nicht gesperrt[18], werden auch noch die Warn- und SOC-Grenzen, der Sollwert und die Rezeptwerte gelesen.

Liegt im Tag ein Alarm vor, wird dieser der internen Collection **`m_cAlarms`** zugefügt und das Flag **`bHasAlarm`** gesetzt.

Nachdem die Schleife beendet ist, werden die Recordsets geschlossen und zerstört.

Nun wird das Event **`intRefresh(pmsRefreshTags)`** erzeugt. Wurden Alarme gefunden, wird auch noch das Event **`intRefresh(pmsRefreshAlarms)`** erzeugt. Außerdem wird noch die interne Prozedur **`AddAlarmsToHistory()`** aufgerufen, die die Alarme in die Historie einbaut.

4.4.5.8 Public Function DoSend() As Boolean

Parameter: *Keine*

Diese Funktion ist dazu gedacht, Änderungen in einzelnen oder mehreren Tags an das PLS zu senden. Zurzeit ist sie jedoch noch nicht implementiert und produziert daher nur den VB-Fehler **`pmsErrorNotYetImplemented`**.

4.4.5.9 Public Function EnableActualizing() As Boolean

Parameter: *ByVal lTagFirst As Long*
Optional ByVal lTagLast As Long = -1

4.4.5.10 Public Function EnableAlarms() As Boolean

Parameter: *ByVal lTagFirst As Long*
Optional ByVal lTagLast As Long = -1

Diese Funktion aktiviert die angegebenen Tags. Das bedeutet, dass diese Tags ab dem nächsten **`DoRefresh()`** aktualisiert werden.

Wird **`lTagFirst`** nicht angegeben, so wird dafür die erste bekannte Tagnummer verwendet. Wird **`lTagLast`** nicht angegeben, wird die letzte bekannte Tagnummer verwendet.

Nun werden alle Tags, deren Nummern im so definierten Bereich liegen, aktiviert.

Wurde mindestens ein Tag aktiviert, liefert die Funktion **`True`**, sonst **`False`**.

Am Ende der Funktion wird **`SaveEnabledTags()`** aufgerufen, das ein internes Array der aktivierten Tags bildet. Dieses wird benötigt, wenn der Server gewechselt wird, da ansonsten alle bisherigen Tags als *nicht aktiviert* bestehen bleiben würden.

[18] Ein Tag ist dann gesperrt, wenn es gerade bearbeitet oder gesendet wird. In diesem Fall dürfen die Grenzwerte nicht verändert werden.

4.4.5.11 Public Function GetConnectionString() As String

Parameter: Optional ByVal lIdx As Long = -1

Diese Funktion liefert den Verbindungstext für einen bestimmten Index. Wird **`lIdx`** nicht angegeben, so wird der aktuelle Index verwendet. Sie wird auch von der Eigenschaft **`ConnectionString`** verwendet.

Der hier gelieferte Verbindungstext beinhaltet nur die drei Elemente **„HST“**, **„USR“** und **„PWD“**. Alle anderen Teile sind entweder nicht bekannt (z.B. **„TYP“**) oder für Wonderware nicht relevant (z.B. **„PRT“**).

4.4.5.12 Public Function GetName() As String

Parameter: ByVal lTagNo As Long

Liefert den Tagnamen des Tags mit der Nummer **`lTagNo`**. Hierbei werden Prä- und Postfixe (z.B. **„DMT_“** und **„_PV“**) entfernt. Wurde das Tag nicht gefunden, liefert die Funktion einen Leerstring zurück.

Beispiel: Das Tag mit dem internen Namen **„FSK_F73186_4_PV“** liefert den Namen **„F73186_4“**.

4.4.5.13 Public Function GetText() As String

Parameter: ByVal lTagNo As Long

Diese Funktion ist dazu gedacht, den Inhalt von Stringtags zu liefern, die (über **`[wwTags]`**) den Tags zugeordnet sind. Zurzeit (06.2004) liefert sie ausschließlich einen Leerstring, da noch keine Stringtags festgelegt sind.

4.4.5.14 Public Function GetValue() As Double

Parameter: ByVal lTagNo As Long

Liefert den aktuell anliegenden Messwert dieses Tags. Dabei ist es gleichgültig, ob das Tag aktualisierend ist, oder nicht, der Wert wird auf jeden Fall aus der Datenbank gelesen.

Kann der Wert nicht gelesen werden, wird als Ergebnis **`-1E+38`** geliefert.

4.4.5.15 Public Function Init() As Boolean

Parameter: Keine

Diese Funktion initialisiert die Verbindung mit der Wonderware-Datenbank. Dazu muss mindestens ein gültiger Verbindungsstring angegeben sein. Ist dies nicht der Fall, erzeugt die Funktion einen Fehler **`pmsErrorInvalidParams`** (**`„invalid or missing connection string“`**).

Eine eventuell bereits bestehende Verbindung wird geschlossen und direkt anschließend eine neue aufgebaut.

Zum Schluss wird die Collection der Tags grundinitialisiert, wodurch alle Tageinstellungen, speziell der Aktualisierungsstatus, verloren gehen.

4.4.5.16 Public Function PrepSendLimits() As Boolean

Parameter: *ByVal lTagNo As Long*
ByVal dblMin1 As Double
ByVal dblMin2 As Double
ByVal dblMax1 As Double
ByVal dblMax2 As Double
Optional ByVal dblSetPoint As Double = g_cdInvalid
Optional ByVal lRecType As Long = -1

Diese Funktion ist dazu gedacht, ein Tag zum Senden der Grenzwerte an das PLS zu präparieren.

Zurzeit ist die Methode jedoch noch nicht implementiert, so dass sie nur einen Fehler des Typs **`pmsErrorNotYetImplemented`** generiert.

4.4.5.17 Public Function PrepSendRecipe() As Boolean

Parameter: *ByVal lTagNo As Long*
ByVal dblMin1 As Double
ByVal dblMin2 As Double
ByVal dblMax1 As Double
ByVal dblMax2 As Double
Optional ByVal dblSetPoint As Double = g_cdInvalid
Optional ByVal lRecType As Long = -1

Diese Funktion ist dazu gedacht, ein Tag zum Senden der Rezeptwerte an das PLS zu präparieren.

Zurzeit ist die Methode jedoch noch nicht implementiert, so dass sie nur einen Fehler des Typs **`pmsErrorNotYetImplemented`** generiert.

4.4.5.18 Public Function ReadAlarmHistory() As Boolean

Parameter: *Keine*

Diese Funktion ist nur als Dummy vorhanden, der immer True als Ergebnis liefert. Bei Wonderware wird die Alarmhistorie nicht auf dem Server, sondern nur in der PMS-Datenbank gehalten, so dass diese Funktion sinnlos geworden ist.

4.4.5.19 Public Function ReadArchiveValues() As Collection

Parameter: *ByVal lTagNum As Long*
ByVal dFrom As Date
ByVal dTo As Date
ByVal arcType As Long

Liefert eine Collection der Archivwerte eines Tags **`lTagNum`** über den Zeitraum zwischen **`dFrom`** und **`dTo`**. Der Parameter **`arcType`** muss zwar angegeben werden, wird aber intern nicht benötigt, da bei Wonderware nicht zwischen Stör- und Archivwerten unterschieden wird. Hierbei werden alle Messungen, die zwischen den beiden Zeitpunkten durchgeführt wurden, geliefert.

Wenn **`dTo`** mit **`0`** angegeben wird, verwendet die Funktion den aktuellen Zeitpunkt. Für **`dFrom`** muss aber ein sinnvoller Zeitpunkt, der vor **`dTo`** liegt, angegeben werden.

Das Ergebnis ist eine Collection mit Tags, die sich durch Referenznummer, Zeitpunkt und Wert unterscheiden. Die Referenznummer ist in diesem Fall nicht die Tagnummer, sondern eine laufende Nummer, beginnend bei **`0`**.

Die Größe der generierten Collection hängt vom Abstand der beiden Zeitpunkte, sowie der Messfrequenz des Tags ab.

4.4.5.20 Public Function ReadData() As Boolean

Parameter: *ByRef tag As clsTag*
Optional bQuick As Boolean = False

Diese Funktion führt den gleichen Vorgang aus wie **DoRefresh()**, jedoch nur für das angegebene Tag. Im Parameter **tag** muss daher bereits die Tagnummer eingestellt sein.

Der Parameter **bQuick** hat dieselbe Bedeutung wie bei **DoRefresh()**. Der einzige Unterschied ist der, dass das Tag vorher nicht aktiviert werden muss.

4.4.5.21 Public Function ReadSOCMinMax() As Collection

Parameter: *ByVal lTagNum As Long*
ByVal dFrom As Date
Optional ByVal dTo As Date = 0

Diese Funktion liefert eine Collection mit vier Werten, die als Variant geliefert werden:

1. dem maximalen Wert aus dem angegebenen Bereich.
2. dem minimalen Wert aus dem angegebenen Bereich.
3. dem durchschnittlichen Wert aus dem angegebenen Bereich.
4. der Anzahl der Werte, aus denen die ersten berechnet wurden.

Der Parameter **lTagNum** ist die Tagnummer des gesuchten Tags, **dFrom** und **dTo** sind die Zeitpunkte, zwischen denen die Daten ermittelt werden sollen, wobei **dTo** optional ist. Wird der Parameter nicht angegeben, wird der aktuelle Zeitpunkt verwendet.

4.4.5.22 Public Function ReConnect() As Boolean

Parameter: *Keine*

In dieser Funktion wird die Verbindung zur Wonderware-Datenbank hergestellt. Ebenfalls wird über **LoadTagRelations()** die Tabelle **m_rsTagRel** mit den Zusammenhängen der unterschiedlichen Tags aus der PMS-Datenbank geladen.

Zuerst wird kontrolliert, ob ein Verbindungsstring vorhanden ist. Wenn nicht, wird ein Fehler **pmsErrorInvalidParams** erzeugt.

Dann wird die Tabelle der Zusammenhänge (**m_rsTagRel**) gelöscht und zerstört.

Als nächstes wird die Verbindung zur Wonderware-Datenbank erstellt, falls sie noch nicht existiert und geschlossen, falls sie schon geöffnet war.

Nun wird in einer Schleife versucht, eine funktionierende Verbindung zu dem, bzw. einem der Wonderware-Server zu erstellen. Dazu wird — beim aktuellen Index **m_lCurrConnection** beginnend — versucht, eine Verbindung mit den Daten herzustellen. Um diesen Vorgang bei Misserfolg nicht allzu lange hinauszuzögern, wird hier ein Verbindungs-Timeout von 5 Sekunden festgelegt.

Hat der Verbindungsaufbau funktioniert, wird **m_rsTagRel** wieder geladen, der aktuelle Index gespeichert und danach das Event **intConnection** mit dem Namen des Servers als Parameter generiert. Außerdem wird der Rückgabewert der Funktion auf **True** gesetzt, um das erfolgreiche Verbinden mitzuteilen.

Schlug die Verbindung dagegen fehl, hängt das weitere Vorgehen davon ab, ob mehrere Verbindungssets vorhanden sind, oder nicht.

Wurde nur ein einziges Set angegeben oder ist **m_bLockChanges** gesetzt[19], wird hier ein Fehler **pmsErrorCantConnect** erzeugt, da die Verbindung nicht hergestellt wurde und auch keine Alternativen vorhanden sind. In diesem Fall wird auch **m_bLockChanges** gelöscht, um den normalen Zustand der Klasse wieder herzustellen.

Sind dagegen mehrere Sets vorhanden und das interne Flag **m_bLockChanges** nicht gesetzt, so wird jetzt über das Event **intWarning(pmsErrorCantConnect)** eine Warnung generiert, dass der Servername, der ebenfalls als Eventparameter mitgeliefert wird, nicht erreichbar ist. Danach wird der interne Index erhöht, um das nächste Set zu testen. Wird dabei der Index, bei dem die Suche begonnen hat, wieder erreicht, wird die Schleife abgebrochen. Der Rückgabewert steht in diesem Fall auf **False**, wodurch der Aufrufer mitgeteilt bekommt, dass keine Verbindung hergestellt werden konnte.

4.4.5.23 Public Function StatisticGenerate() As Boolean

Parameter: *ByVal dFrom As Date*
ByVal dTo As Date

Ermittelt statistische Werte für die angegebene Zeitdauer. Die ermittelten Statistikwerte sind **Durchschnitt**, **Minimum**, **Maximum**, **Anzahl**, **Startzeitpunkt** und **Endezeitpunkt** und werden direkt in die jeweiligen Tags eingetragen.

Damit die Werte ermittelt werden können, muss vor dem Aufruf dieser Methode das **Action**-Flag der gewünschten Tags auf **tacGetStatistic** gestellt werden.

Ähnlich wie bei **DoRefresh()** werden hier die statistischen Werte aller Tags, bei denen das Flag **tacGetStatistic** gesetzt ist, ermittelt und in den Statistikfeldern **…DateStart**, **…DateEnd**, **…Count**, **…Average**, **…Minimum** und **…Maximum** des jeweiligen Tags eingetragen. Das Flag **tacGetStatistic** wird dabei nicht gelöscht, d.h. wenn die Funktion erneut aufgerufen wird, werden die Werte für die gleichen Tags erneut ermittelt.

4.4.5.24 Public Function TagRange() As Boolean

Parameter: *ByVal lBegin As Long*
ByVal lEnd As Long

Diese Funktion war ursprünglich dazu gedacht, den Bereich der zu aktualisierenden Tags festzulegen. Jedoch werden die Einstellungen, die hiermit festgelegt werden, nur noch als Defaultwerte bei **Disable**- / **EnableActualizing()**, sowie **Disable**- / **EnableAlarms()** verwendet.

4.4.5.25 Public Function ValidData() As Boolean

Parameter: *Keine*

Diese Funktion prüft, ob mindestens ein Verbindungsset vorhanden ist. Es wird dabei nicht getestet, ob es auch tatsächlich möglich ist, damit eine Verbindung herzustellen.

[19] Das Flag wird in der Funktion **ChangeServer()** gesetzt, um das automatische Suchen gültiger Verbindungen zu unterdrücken.

4.4.6 Öffentliche Eigenschaften[20]

Zur Beachtung: Die Eigenschaften, die sich auf Alarme beziehen, werden zwar noch unterstützt, jedoch nicht mehr verwendet, da die gesamte Alarmverwaltung komplett in das Programm **`PMS_DbAlarm.exe`** verlagert wurde.

4.4.6.1 Public Property Get AlarmActual() As Collection

Liefert eine Collection der aktuell anliegenden Alarme. Sollte die interne Collection noch nicht initialisiert sein, wird sie hier initialisiert.

4.4.6.2 Public Property Get AlarmCount() As Long

Liefert die Anzahl der aktuell anliegenden Alarme in der internen Collection.

4.4.6.3 Public Property Get AlarmHist() As Collection

Liefert eine Collection aller Alarme, die seit dem Start des Moduls aufgelaufen sind. Sollte die interne Collection noch nicht initialisiert sein, wird sie hier initialisiert.

4.4.6.4 Public Property Get AlarmHistCount() As Long

Liefert die Anzahl aller Alarme in der internen Collection der Alarmhistorie.

4.4.6.5 Public Property Get AlarmHistIndex() As Long

Liefert den Index, ab welchem die Alarmhistorie bei der nächsten Abfrage gelesen wird.

4.4.6.6 Public Property Let AlarmHistIndex()

Parameter: *ByVal lIndex As Long*

Legt den Index fest, ab welchem die Alarmhistorie bei der nächsten Abfrage gelesen wird.

4.4.6.7 Public Property Get AlarmHistSize() As Long

Liefert die Größe des Alarmfensters. Dies ist die Anzahl der historischen Alarme, die maximal gespeichert werden. Standardmäßig ist dieser Wert auf 1000 eingestellt, er kann jedoch auch verändert werden.

4.4.6.8 Public Property Let AlarmHistSize()

Parameter: *lNewSize As Long*

Mit dieser Eigenschaft wird die Größe des Alarmfensters eingestellt. Im Normalfall ist der Standardwert von 1000 jedoch ein guter Kompromiss.

[20] Auch hier bedeutet „Öffentlich“ nur, dass die Prozeduren, Funktionen, Eigenschaften, etc. in der kompletten DLL sichtbar sind, nicht aber, dass sie über das COM-Interface freigegeben sind!

4.4.6.9 Public Property Get AlarmHistUpdate() As Collection

Liefert in einer Collection die neuen Alarme seit dem letzten Abruf und merkt sich die zuletzt gelesene Position in der internen Variable **`m_lAlarmHistLastRead`**.

4.4.6.10 Public Property Get Betrieb() As String

Liefert das aktuell eingestellte Kürzel für den Betrieb (**„P2K“**, **„DMT“**, etc.).

4.4.6.11 Public Property Let Betrieb()

Parameter: ByVal sBetrieb As String

Legt den aktuellen Betrieb fest. Hat sich der neue gegenüber dem vorherigen geändert, so wird die Klasse mit **`Init()`** neu initialisiert, um die entsprechenden Tags richtig zu stellen.

4.4.6.12 Public Property Get ConnectionSetCount() As Long

Liefert die Anzahl der eingestellten Verbindungssätze. Wurde **`ConnectionString`** noch nicht aufgerufen, ist dies immer **`0`**.

4.4.6.13 Public Property Get ConnectionSetIndex() As Long

Liefert den Index des aktuellen Verbindungssatzes. Existiert mindestens ein Verbindungssatz, so wird der Index 1-basiert geliefert. Ansonsten liefert die Eigenschaft **`0`**.

4.4.6.14 Public Property Let ConnectionSetIndex()

Parameter: ByVal lNewIndex As Long

Legt den Index des aktuellen Verbindungssatzes fest. Der Wert ist 1-basiert und darf nicht größer sein als die Anzahl der Verbindungssätze.

4.4.6.15 Public Property Get ConnectionString() As String

Liefert den Verbindungsstring für die aktuelle Verbindung. Wurde noch kein Verbindungsstring definiert, so liefert diese Eigenschaft einen Leerstring.

4.4.6.16 Public Property Let ConnectionString()

Parameter: ByVal sNewConnectionString As String

Wie schon in der Klasse **`PMSKopplung`**, Eigenschaft **`ConnectionString`** (siehe Seite 24) ausführlich dargestellt, werden mit dieser Eigenschaft der oder die Verbindungssätze festgelegt.

Für Wonderware sind nur **„SRV“**, **„USR“**, **„PWD“** und **„BTR“** relevant. Alle anderen Werte werden ignoriert.

4.4.6.17 Public Property Get EnabledTags() As Collection

Liefert eine Collection, die ausschließlich die aktivierten Tags enthält.

4.4.6.18 Public Property Get Hostname() As String

Liefert den aktuell verwendeten Servernamen. Ist noch kein Verbindungssatz definiert, oder noch keine Verbindung aufgebaut, so wird ein Leerstring geliefert.

Beim forcierten oder impliziten Wechsel des Servers kann sich diese Eigenschaft natürlich ändern.

4.4.6.19 Public Property Get Password() As String

Liefert das aktuell verwendete Kennwort. Ist noch kein Verbindungssatz definiert, oder noch keine Verbindung aufgebaut, so wird ein Leerstring geliefert.

Beim forcierten oder impliziten Wechsel des Servers kann sich diese Eigenschaft natürlich ändern.

4.4.6.20 Public Property Get Port() As Variant

Liefert immer den Wert **-1**, da bei Wonderware kein Port verwendet wird.

4.4.6.21 Public Property Get RefreshTime() As Long

Liefert den maximalen Wert, der für **[StorageRate]** beim aktuellen Betrieb definiert ist. Ist noch kein Betrieb festgelegt, so wird ein fester Wert von **10** zurückgegeben.

4.4.6.22 Public Property Get TagBegin() As Long

Liefert den Anfang des Bereiches der zu aktualisierenden Tags. (Wird auch durch die Funktion **TagRange()** festgelegt).

4.4.6.23 Public Property Let TagBegin()

Parameter: *ByVal lBegin As Long*

Legt den Anfang des Bereiches der zu aktualisierenden Tags fest. (Wird auch durch die Funktion **TagRange()** festgelegt).

4.4.6.24 Public Property Get TagCount() As Long

Liefert die Anzahl der auszuwertenden Tags. Dies ist einfach der Wert, der sich aus **TagEnd – TagBegin + 1** berechnet.

4.4.6.25 Public Property Let TagCount()

Parameter: *ByVal lAnz As Long*

Stellt die Anzahl der auszuwertenden Tags ein. Dadurch wird einfach der neue Wert von **TagEnd** aus **TagBegin + lAnz - 1** errechnet.

4.4.6.26 Public Property Get TagEnd() As Long

Liefert das Ende des Bereiches der zu aktualisierenden Tags. (Wird auch durch die Funktion **`TagRange()`** festgelegt).

4.4.6.27 Public Property Let TagEnd()

Parameter: ByVal lEnd As Long

Legt das Ende des Bereiches der zu aktualisierenden Tags fest. (Wird auch durch die Funktion **`TagRange()`** festgelegt).

4.4.6.28 Public Property Get TagLast() As Long

Liefert die höchste Tagnummer des aktuellen Betriebes. Ist noch kein Betrieb festgelegt oder das System noch nicht initialisiert, wird der feste Wert **`32768`** geliefert.

4.4.6.29 Public Property Get TagRecordset() As ADODB.Recordset

Diese Eigenschaft erstellt ein Recordset, das mit allen Tags der internen Liste gefüllt wird. Alle Felder sind vom Typ **`adVarChar(255)`**, also Strings mit maximal 255 Zeichen. Die Felder haben die gleichen Bezeichnungen wie die entsprechenden Eigenschaften der Klasse **`clsTag`**:

Feldname	Feldname
`Ref`	`RecipeSetPoint`
`Name`	`RecipeLimitMin2`
`FilteredName`	`RecipeLimitMin1`
`ID`	`RecipeLimitMax1`
`Description`	`RecipeLimitMax2`
`Unit`	`State`
`Func`	`Action`
`DBA`	`RecordType`
`DBE`	`StatisticAverage`
`Value`	`StatisticMinimum`
`TagText`	`StatisticMaximum`
`SetPoint`	`StatisticCount`
`LimitMin2`	`StatisticDateStart`
`LimitMin1`	`StatisticDateEnd`
`LimitMax1`	`LastUpdate`
`LimitMax2`	

Tabelle 22: Recordset-Felder

Das gelieferte Recordset ist „disconnected“, also nicht an eine Datenbank gebunden. Es enthält *alle* Tags der Klasse, unabhängig davon, ob sie aktualisierend sind oder nicht. Ist die Klasse noch nicht initialisiert, so ist das Recordset leer (enthält also keine Records).

4.4.6.30 Public Property Get Tags() As Collection

Ähnlich wie bei **`TagRecordset()`** werden hier alle Tags der Klasse geliefert, jedoch in Form einer Collection. Ist die Klasse noch nicht initialisiert, so wird eine leere Collection geliefert.

4.4.6.31 Public Property Get Username() As String

Liefert den aktuell verwendeten Benutzernamen. Ist noch kein Verbindungssatz definiert, oder noch keine Verbindung aufgebaut, so wird ein Leerstring geliefert.

Beim forcierten oder impliziten Wechsel des Servers kann sich diese Eigenschaft natürlich ändern.

4.4.7 Öffentliche Ereignisse

Die hier erzeugten Ereignisse gelangen nicht bis ins Client-Programm, sondern nur bis zur Klasse **PMSKopplung**. Diese leitet sie dann über die entsprechenden eigenen Ereignisse an das eigentliche Programm weiter. Das ist prinzipiell genau die gleiche Methode wie bei den Prozeduren und Funktionen.

4.4.7.1 Public Event intConnection()

Parameter: *sHost As String*

Dieses Event wird nach dem erfolgreichen Aufbau der Verbindung erzeugt. Als Parameter wird der Name des Servers geliefert, mit dem sich die Klasse tatsächlich verbunden hat.

4.4.7.2 Public Event intRefresh()

Parameter: *RefreshType As Long*

Dieses Event wird jedes Mal erzeugt, wenn sich die internen Daten geändert haben. Der Parameter **RefreshType** kann einen der folgenden Werte enthalten:

pmsRefreshTags:	Wird erzeugt, nachdem mit der Funktion **DoRefresh()** neue Werte gelesen wurden. Dies ist das wichtigste Event, da zu diesem Zeitpunkt alle Tags ihre aktuellen Werte zugewiesen bekommen haben.
pmsRefreshStatistics:	Wird am Ende der Funktion **StatisticGenerate()** erzeugt, nachdem die relevanten Statistikwerte in die Tags eingetragen wurden.
pmsRefreshAlarms:	Wird erzeugt, wenn beim Lesen neuer Daten in **DoRefresh()** Alarme festgestellt wurden.
pmsRefreshAlarmUpdate:	Wird erzeugt, wenn beim Lesen neuer Daten in **DoRefresh()** Alarme festgestellt wurden, die sich von den vorherigen unterscheiden.
pmsRefreshSend:	Wird von der Funktion **DoSend()** verschickt, sobald das Senden von Werten abgeschlossen ist. Da diese Funktion zurzeit noch nicht implementiert ist, wird dieser Wert auch noch nicht gesendet.

4.4.7.3 Public Event intWarning()

Parameter: *Warningcode As Long*
sInfo As String

Dieses Event wird beim Auftreten von nicht-fatalen Schwierigkeiten erzeugt. Der Parameter **`Warningcode`** kann dabei einen der folgenden Werte enthalten:

`pmsErrorCantConnect`: Wird von der Funktion **`ReConnect()`** erzeugt, wenn mehr als ein Verbindungssatz vorhanden ist und die Verbindungsaufnahme mit einem davon fehlgeschlagen ist.
`sInfo` enthält dann den Text **`"no connection to <HOST>"`**

`pmsErrorConnectionLost`: Wird von der Funktion **`CheckConnection()`** erzeugt, wenn keine Verbindung zur Wonderware-Datenbank besteht.
`sInfo` enthält dann den Text **`"connection lost"`**