Software-Dokumentation

PMS_Transfer - Datentransfer mit SAP

erstellt von Klaus Raithel

Version 216 (13.01.04)
(in Arbeit)

KoSA

COMPUTER OUTFIT

Inhalt

Abbildungen

Tabellen

1 Allgemeines

Die Daten, die in den einzelnen Betrieben von PMS erfasst werden, müssen — zum Teil — an SAP gesendet werden, um die dort ablaufenden buchhalterischen Vorgänge zu unterstützen, bzw. erst möglich zu machen (z.B. Chargendaten und Messwerte).

Andererseits benötigt PMS auch Daten, die in SAP generiert werden, um seine eigenen Daten abgleichen zu können (z.B. Aufträge, Materialnummern und Typen).

Da weder PMS in der Lage ist, direkt in SAP Daten einzutragen, noch SAP Zugriff auf die Interna von PMS hat, musste eine Schnittstelle geschaffen werden, die eine Kommunikation zwischen den beiden grundlegend unterschiedlichen Systemen ermöglicht.

Da PMS Microsoft SQL Server verwendet, SAP jedoch eine Oracle-Datenbank, ist es notwendig, dass der Computer, auf dem **PMS_Transfer** läuft, über diese beiden Schnittstellen verfügt. Die Schnittstelle zur PMS-Datenbank wird durch MDAC 2.7, dessen Vorhandensein bei der Installation des PMS-Basispakets sichergestellt wird, installiert. Die Schnittstelle zu Oracle muss getrennt installiert werden.

1.1 Transfer

Um den Datentransfer komplett zu automatisieren, läuft auf einem Computer das Programm **PMS_Transfer**. Dieses Programm führt in regelmäßigen Abständen Abfragen im der PMS- und in der SAP-Datenbank durch, ob Daten zu übertragen sind. Wenn ja, werden die entsprechenden Daten in das passende Format des Zielsystems konvertiert und eingetragen.

PMS_Transfer darf im gesamten System nur ein einziges Mal laufen, da sonst Doppelsendungen in beiden Richtungen vorkommen könnten. Aus diesem Grund (und zur Überwachung) ist hier die Klasse **CSystemStatus** eingebaut, die unter anderem kontrolliert, ob das Programm im System schon auf irgendeinem Rechner läuft. Wenn ja, wird der Start verweigert.

Das Programm läuft normalerweise minimiert, d.h. es wird nur als ein Symbol  im Systemtray angezeigt. Durch Doppelklick auf das Symbol wird es angezeigt.

1.2 Ablauf des Programms

Das Programm läuft in einer timergesteuerten Schleife (Laufzeit: 1 Sekunde). In der Behandlungsroutine wird zuerst geprüft, ob der nächste Zeitpunkt zum Aufrufen eines Jobs bereits erreicht ist. Wenn nicht, wird die Timerroutine sofort verlassen. Sonst wird die Jobbehandlung (**CheckJobs()**) ausgeführt.

Darin wird als erstes geprüft, ob die beiden Verbindungen zu den Datenbanken funktionsfähig sind. Bei Problemen wird hier auch versucht, die jeweilige Verbindung wieder herzustellen. Schlägt auch das fehl, so wird ein Fehlerzähler um eins erhöht und die Routine abgebrochen, indem zum Label **Hell**: verzweigt wird. Trat dieser Fehler zehnmal hintereinander auf, so wird der Fehler in der PMS-Tabelle **[Int_Errorlog]** eingetragen[1] und das Programm komplett beendet.

[1] natürlich nur dann, wenn die PMS-Datenbank erreichbar ist.

Trat bei den Prüfungen kein Fehler auf, so wird geprüft, ob ein so genannter „forcierter Job“ vorliegt. Diese können durch die Stammdatenverwaltung erzeugt werden, um bspw. das Herunterladen von Aufträgen sofort zu starten. Ist ein solcher forcierter Job vorhanden, so wird die Startzeit dieses Jobs auf „Jetzt“ gesetzt und der aktuelle Index der Jobverarbeitung auf diesen Job gestellt. Weiterhin wird der forcierte Job in der PMS-Datenbank auf „erledigt“ gestellt.

Unabhängig davon, ob ein Job forciert wurde oder nicht, wird nun der nächste Job in der Reihenfolge ausgeführt. Sofern die interne Startzeit dieses Jobs noch nicht erreicht ist, kehrt die Prozedur sofort wieder zurück. Ansonsten führt der Job seine Aufgabe aus und kehrt danach erst zurück. Nach kleineren Aufräumarbeiten wird die Funktion beendet und kehrt zurück in die Timerbehandlung. Dort wird jetzt noch der nächste Zeitpunkt zur Ausführung berechnet und das Programm wartet auf das nächste Timerereignis.

1.3 Sicherstellen des Programmlaufs

Da der SAP-Transfer ein wichtiger Faktor im allgemeinen Ablauf von PMS ist, muss sichergestellt sein, dass der Prozess immer läuft. Üblicherweise würde ein solcher Prozess als Dienst konzipiert, jedoch wird andererseits verlangt, dass die Programme in Visual Basic geschrieben sind. Es existiert zwar ein Modul, um VB-Programme als Dienst zu starten, jedoch empfiehlt der Hersteller (Microsoft®), dieses mehr als „wissenschaftliche Spielerei“ denn als sinnvolle Erweiterung zu sehen. Beispielsweise ist das Debuggen eines derartigen Dienstes nur in einem Emulationsmodus möglich.

Da diese Einschränkungen so nicht hingenommen werden können, wurde das Programm als normales EXE-Modul realisiert[2]. Um den permanenten Lauf zu garantieren, sorgt eine einfache Batchdatei dafür, dass das Programm nach einem eventuellen Beenden sofort wieder angestoßen wird.

```
@echo off

title Keep Transfer

:Loop
start /min PMS_Transfer.exe
techo -fPMS_Logging.txt SAP-Transfer wurde gestartet
WaitFor "PMS SAP-Transfer"
techo -fPMS_Logging.txt SAP-Transfer wurde beendet!

goto Loop
```

Tabelle 1: Batch KeepTransfer.cmd

Diese Batchdatei verwendet außer **PMS_Transfer** noch die beiden einfachen Programme **TECHO** (siehe Seite 74) und **WAITFOR** (siehe Seite 75).

Zuerst wird der SAP-Transfer gestartet; durch die Verwendung von **START** wartet das Batch nicht auf das Beenden des Programms. Danach wird mit **TECHO** der Zeitpunkt und der Start-Text in die Protokolldatei PMS_Logging.txt eingetragen und mit **WAITFOR** auf das Beenden des Hauptprogramms gewartet. Sobald dies — aus welchen Gründen auch immer — passiert, wird der Zeitpunkt und der Ende-Text ebenfalls in die Protokolldatei eingetragen. Anschließend beginnt der Aufrufzyklus wieder von vorn.

[2] Es wurde auch versucht, dieses Programm dann durch Hilfsmittel wie „SRVANY“ als Dienst zu starten. Jedoch schlugen diese Versuche allesamt fehl. Offenbar ist die Nachrichtenverwaltung in Visual Basic doch nicht ganz so, wie es von einem Windows-Programm erwartet wird.

Konsequenterweise bedeutet dies, dass — wenn das Programm wirklich beendet werden soll — zuerst die Batchdatei beendet werden muss, bevor das Programm beendet wird.

1.4 Kommunikation

Die Kommunikation wird über spezielle Tabellen in der SAP-Datenbank realisiert (so genannte Z-Tabellen, da sie alle mit Z beginnen). Die zurzeit verwendeten Tabellen sind:

Tabellenname	*Richtung*[3]	*Beschreibung*
ZAKTION_VP_4B[4]	U & D	Wird als „Triggertabelle“ verwendet. Nachdem Daten in eine der anderen Tabellen eingefügt wurden, wird hier ein spezieller Datensatz eingetragen, der SAP darüber informiert, dass nun Daten abzuholen sind. Weiterhin werden hier Anforderungen eingetragen, die SAP dazu auffordern, neue Daten in einer der anderen Tabellen einzutragen.
ZAUF_VP_4B	D	In dieser Tabelle werden neue Aufträge von SAP übertragen.
ZCHARGE_VP_4B	U	In dieser Tabelle werden fertig gestellte Chargen an SAP gemeldet.
ZMESSWERT_VP_4B	U	In dieser Tabelle werden Messwerte, die zu einer Charge ermittelt wurden, an SAP übertragen.
ZQM_BATCH_VP_4B	D	In dieser Tabelle liefert SAP QM-Daten.
ZQM_LIEFER_VP_4B	D	In dieser Tabelle sendet SAP Lieferdaten.
ZSPEZ_VP_4B	D	In dieser Tabelle sendet SAP Spezifikationen.

Tabelle 2: Kommunikationstabellen

Welche Tabelle für welche Daten verwendet wird, hängt von den SQL-Kommandos ab, die in den einzelnen Jobs verwendet werden.

[3] U = Upload (PMS **è** SAP), D = Download (SAP **è** PMS)

[4] 4B ist die Bezeichnung für das Werk „KoSa Offenbach“

2 Module, Fenster und Klassen

PMS_Transfer besteht aus den folgenden Einzelteilen:

Dateiname	*Modulname*	*Beschreibung*
basMain.bas	**basMain**	Enthält allgemeine Funktionen und Prozeduren, die keinem speziellen Modul zugeordnet werden können.
basSQL.bas	**basSQL**	Enthält alle SQL-Statements (außer den SAP-Kommandos) sowie Methoden zum Formatieren, bzw. Ausfüllen derselben. Außerdem sind hier noch die Routinen zum Verbindungsaufbau mit der PMS-Datenbank zu finden.
basSAP.bas	**basSAP**	Enthält alle SAP-Kommandos sowie Methoden zum Formatieren, bzw. Ausfüllen derselben. Außerdem sind hier noch die Routinen zum Verbindungsaufbau mit der SAP-Datenbank zu finden.
MultiHook.bas	**MultiHook**	Dieses Modul ermöglicht die Überwachung normaler Windows-Nachrichten. Es wird in **`frmMain`** verwendet, um die Größenänderung des Fensters zu begrenzen.
frmMain.frm	**frmMain**	Enthält das Hauptfenster des Programms.
dlgMsg.frm	**dlgMsg**	Dialog, der angezeigt wird, wenn Fehler auftreten. Da das Programm meist unbeaufsichtigt laufen soll, schließt sich dieser Dialog nach 5 Sekunden selbstständig.
dlgUserInfo.frm	**dlgLoginPMS**	Dialog, um die Anmeldeinformationen für die PMS-Datenbank einzugeben.
dlgSAP.frm	**dlgLoginSAP**	Dialog, um die Anmeldeinformationen für die SAP-Datenbank einzugeben.
dlgInfo.frm	**dlgInfo**	Hilfefenster mit einigen Informationen zu den Einstellungen.
CJob.cls	**CJob**	Klasse zur Implementation der Jobs.
CSapCharge.cls	**CSapCharge**	Wird verwendet, um Chargendaten an SAP zu senden.
CSapGlobals.cls	**CSapGlobals**	Enthält einige SAP-Angaben, sowie eine Methode, um Aufträge in die Aktionstabelle zu senden.
CSapMesswert.cls	**CSapMesswert**	Wird verwendet, um Messwerte an SAP zu senden.
CSystemStatus.cls	**CSystemStatus**	Verwaltet und überwacht den Status des Programms. Identische Kopien dieser Klasse werden auch im Client-Programm, und der Stammdatenverwaltung zur Darstellung der Status verwendet.
CSysTray.cls	**CSysTray**	Klasse zur Anzeige eines Symbols im rechten unteren Bereich (**`SysTray`**) des Windows-Bildschirms. Sie unterstützt auch das Verstecken des Fensters beim Minimieren.

Tabelle 3: Module, Fenster und Klassen

3 Module

In diesem Kapitel werden die Methoden der einzelnen Module (Dateierweiterung **.bas**) beschrieben.

3.1 basMain

basMain enthält allgemeine Routinen, Funktionen und Daten, auf die von allen Modulen aus zugegriffen werden kann.

3.1.1 Globale Variablen

Hier sind die folgenden globalen Variablen definiert:

Variablenname	*Typ*	*Beschreibung*
g_PMSConn	**ADODB.Connection**	Das Connection-Objekt, das für die Kommunikation mit der PMS-Datenbank verwendet wird.
g_SAPConn	**ADODB.Connection**	Das Connection-Objekt, das für die Kommunikation mit der SAP-Datenbank verwendet wird.
g_sPMSConnStr	**String**	Verbindungsstring, der durch den Anmeldedialog definiert und den kompletten Programmlauf behalten wird.
g_SAP	**CSapGlobals**	Enthält einige SAP-Angaben, sowie eine Methode, um Aufträge in die Aktionstabelle zu senden.
g_SysStatus	**CSystemStatus**	Dieses Objekt dient dazu, den Status der Partieeröffnung in der Datenbank zu aktualisieren (siehe Klasse **CSystemStatus**)

Tabelle 4: Globale Variablen

3.1.2 Konstanten

Die in diesem Bereich definierten Konstanten sind feste Einstellungen, die während des Programmlaufs nicht geändert werden (können!)

Der erste Abschnitt beinhaltet Konstanten für die SAP-Anmeldung. Dies sind Defaultwerte, die dann verwendet werden, wenn in der Datenbank (Tabelle **[Int_Parameter]**) noch keine Angaben eingetragen sind.

Danach folgen Defaultwerte, die Namen der bisher bekannten Betriebe, sowie Konstanten zur Verwendung mit und die Deklaration der entsprechenden Standard-Funktion (**SendMessage()**).

3.1.2.1 Defaultwerte für SAP

Konstantenname	*Typ*	*Inhalt*	*Beschreibung*
g_sSAPMandant	**String**	"450"	Mandantennummer zur Anmeldung an SAP
g_sSAPWerk	**String**	"4B"	Kennnummer des Werks (KoSa Offenbach)
g_sSAPUser	**String**	"SAP Transfer"	Benutzername
g_sSAP_TID	**String**	" " (8 Spaces)	Sinn muss noch ermittelt werden (Transaktions-ID?)

Tabelle 5: SAP-Defaultwerte

3.1.2.2 Konstanten für Registry

Konstantenname	*Typ*	*Inhalt*	*Beschreibung*
g_chRegRoot	**Long**	HKEY_LOCAL_MACHINE	Symbolischer Name für den Registrierungszweig
g_csRegPath	**String**	"Software\KoSa\PMS\Transfer"	Schlüssel, in dem die Angaben für **`PMS_Transfer`** stehen.
g_csTimerDelay	**String**	"Timer delay"	Die Zeit in Sekunden, die zwischen den Aufrufen der Jobs gewartet werden soll
g_csWinLeft	**String**	"Window Left"	Namen der Werte, in denen die Fensterposition und –größe abgelegt wird
g_csWinTop	**String**	"Window Top"	
g_csWinWidth	**String**	"Window Width"	
g_csWinHeight	**String**	"Window Height"	
g_csRegDbPath	**String**	"Software\KoSa\PMS\SQLDB"	Schlüssel, in dem die Angaben für die Datenbank stehen.
g_csRegDbServer	**String**	"DB Server"	Name des Servers
g_csRegDbUser	**String**	"DB User"	Benutzername
g_csRegDbPwd	**String**	"DB Pwd"	Kennwort (verschlüsselt)
g_csRegDbCatalog	**String**	"DB Catalog"	Datenbankname
g_csRegSapPath	**String**	"Software\KoSa\PMS\SAP"	Schlüssel, in dem die Angaben für die SAP-Datenbank stehen.
g_csRegSapTnsName	**String**	"SAP TNSName"	Name der Zugangsdefinition zur Oracle-Datenbank
g_csRegSapUser	**String**	"SAP User"	Benutzername dazu
g_csRegSapPwd	**String**	"SAP Pwd"	Kennwort dazu (verschlüsselt)

Tabelle 6: Registry-Konstanten

3.1.2.3 Sonstige Konstanten

Konstantenname	*Typ*	*Inhalt*	*Beschreibung*
g_csCheckInterval	**String**	"StepTwoInterval"	Name der Variablen in der Tabelle **[Int_Parameter]**, die die Wartezeit in Sekunden zwischen der Anforderung und dem Abholen bei zweistufigen Jobs enthält.
g_csStatusName	**String**	"Transfer"	Bezeichnung, mit der sich das Programm **PMS_Transfer** in der Tabelle **[Status]** anmeldet.
LVSCW_AUTOSIZE_USEHEADER	**Long**	-2	Windows-Konstanten zum Setzen der Spaltenbreite von ListViews.
LVM_SETCOLUMNWIDTH	**Long**	&H101E (4126)	

Tabelle 7: Sonstige Konstanten

3.1.3 Prozeduren

Die folgenden Prozeduren sind global, d.h., sie sind von jedem Modul, Fenster oder Klasse aus erreichbar.

3.1.3.1 Public Sub Main()

Parameter: Keine

Diese Prozedur ist in den Projekteigenschaften als Startpunkt des Programms definiert. Das bedeutet, sie wird vor jedem Fenster, o.ä. aufgerufen. Die folgenden Aufgaben werden hier erledigt:

- Kommandozeilen-Parameter einlesen. Als mögliche Parameter sind die folgenden Texte erlaubt[5]:

  -FORCE Wird dieser Parameter angegeben, so kann das Programm ein zweites Mal gestartet werden, obwohl bereits eine andere Instanz läuft. Dieser Parameter sollte ausschließlich zu Testzwecken verwendet werden, da mehrere SAP-Transfers zu Schwierigkeiten führen können!

  -MIN Dieser Schalter bewirkt, dass sich das Programm sofort nach dem Start in den SysTray-Bereich zurückzieht. Dort wird es dann durch das Symbol **T** dargestellt.

- Eventlogging aktivieren. Dadurch wird es ermöglicht, dass das Programm Einträge im lokalen Eventlog generieren kann.
- SAP-Objekt initialisieren.
- Die Verbindung zur PMS-Datenbank aufbauen. Schlägt dies — auch nach Rückfrage — fehl, so wird das Programm sofort beendet.
- SAP-Defaultwerte aus der PMS-Datenbank, Tabelle **[Int_Parameter]** holen.

[5] Die Parameter werden jeweils durch einen Bindestrich ("-") oder einen Schrägstrich ("/") eingeleitet. Bei allen Parametern ist die Schreibweise (groß/klein) völlig wahlfrei.

- Danach wird das Modul zum Systemstatus initialisiert und gestartet, das den Status des Programms in der Datenbank periodisch aktuell hält. Bei diesem Vorgang wird auch geprüft, ob das Programm für diesen Betrieb — unabhängig vom Rechner — bereits läuft. Wenn ja (und der Schalter **FORCE** nicht angegeben wurde), beendet sich das Programm nach einer Meldung jetzt wieder.
- Als letztes wird das Hauptfenster **frmMain** geladen und — sofern der Parameter **MIN** angegeben wurde — sofort minimiert.

3.1.3.2 Public Sub CloseConnections()

Parameter: Keine

Schließt die Verbindung zur PMS-Datenbank und zur SAP-Datenbank, löscht beide Objekte sowie das allgemeine SAP-Objekt.

3.1.3.3 Public Sub ErrorHandler()

Parameter: ByVal sInfo As String
Optional ByRef vErrCode As Variant

Diese Methode wird immer dann aufgerufen, wenn ein Fehler in einer Funktion oder einer Methode auftrat. Dazu wird — normalerweise zu Beginn der Methode — die Fehlerbehandlung eingeleitet:

```
Public Sub Main()
  On Error GoTo Hell
   ... hier die normalen Anweisungen der Prozedur ausführen ...
Hell:
  ErrorHandler "Main"   ' Fehlerbehandlung aufrufen
End Sub
```

Code 1: Fehlerbehandlung aktivieren

In der Methode wird zuerst versucht, den Fehler so genau wie möglich zu ermitteln. Dazu gehört auch, dass der komplette Fehlerstapel[6] aus der Datenbank abgefragt wird, falls der Fehler aus dieser Richtung kam. Dieser Stapel wird dann in einem String zusammengeführt, der — mit dem Dialog **dlgMsg** (siehe Seite 43) angezeigt wird.

Der Parameter **sInfo** wird als zusätzliche Information in den Text eingebaut.

Ist die Datenbank prinzipiell erreichbar, so wird der angezeigte Text auch noch in der Tabelle [**Int_ErrorLog**] gespeichert.

Außerdem wird dieser Text dann noch im Eventlog des Rechners gespeichert.

[6] In der Datenbank können mehrere Fehler vorliegen; dies sind dann üblicherweise Folgefehler.

3.1.3.4 Public Sub LogError()

Parameter: *ByVal lErrNr As Long*
ByVal sErrSource As String
ByVal sErrLocation As String
ByVal sErrDescription As String
Optional ByRef vErrCode As Variant

Diese Methode speichert den durch die Parameter definierten Fehler in der Datenbanktabelle [**Int_ErrorLog**]. Dabei werden auch zusätzliche Informationen wie Datum und Uhrzeit, sowie der Name des Rechners und der Name des Benutzers gespeichert.

Tritt an dieser Stelle ein Fehler auf, so wird nicht **ErrorHandler** aufgerufen, da dies zu einer Endlosschleife führen würde. Vielmehr wird der Fehler einfach „nach oben“ weitergereicht. Die darüber liegende Methode kümmert sich dann um die weitere Abwicklung.

3.1.3.5 Public Sub LogMessage()

Parameter: *sMessage As String*

Ähnlich wie **LogError** speichert diese Methode die übergebene Nachricht in **sMessage**, jedoch in der Tabelle [**Srv_Serverlog**]. Dabei werden auch zusätzliche Informationen wie Datum und Uhrzeit, sowie der Name des Rechners gespeichert.

3.1.3.6 Public Sub ClearIpcCommand()

Parameter: *ByVal sParam As String*
ByVal tyWhich As atTypes

Diese Prozedur löscht forcierte SAP-Kommandos aus der Tabelle **[IPC]**. Dazu wird der Eintrag so verändert, dass dem Text im Feld **[Val]** der Text „**-OK**“ angehängt und im Feld **[Sender]** der Text „**Transfer**“ eingetragen wird.

Der Parameter **tyWhich** definiert, welches Kommando gelöscht werden soll, **sParam** wird zur genaueren Spezifikation an das SQL-Kommando weitergeleitet und beinhaltet hier immer den Text „**InitTransfer**“. Die möglichen Werte für **tyWhich** sind:

tyWhich	*Kommando*	*Beschreibung*
atTypeOrderGet[7] **atTypeOrderRequest**	**"Aufträge"**	Anfordern von Aufträgen.
atTypeDeliverDataGet atTypeDeliverDataRequest	**"Lieferdaten"**	Anfordern von Lieferdaten.
atTypeQmBatchGet atTypeQmBatchRequest	**"QMDaten"**	Anfordern von QM-Daten.
atTypeSpecGet atTypeSpecRequest	**"Spezifikationen"**	Anfordern von Spezifikationen.

Tabelle 8: IPC-Kommandos

Siehe auch **GetIpcCommand()** auf Seite 11.

[7] Die Kommandos mit „Request“ und „Get“ am Ende sind in diesem Fall synonym, da es keine getrennten Kommandos für Anforderungen und Abholungen gibt.

3.1.4 Funktionen

3.1.4.1 Public Function Num2Str() As String

Parameter: ByVal n As Long

Konvertiert eine Fehlernummer in einen Text. Negative Zahlen werden als Hex-Zahl dargestellt, positive als normale Dezimalzahl. Diese Funktion ist bei Fehlermeldungen interessant, da die Hexzahl (z.B. **80010C00**) einfacher zu interpretieren ist als das dezimale Äquivalent (z.B. **-2147552256**).

3.1.4.2 Public Function Str2Dbl() As Double

Parameter: ByVal s As Variant

Konvertiert Dezimal-Strings in Doublezahlen. Dabei ist es gleichgültig, ob die Zahlen in **s** mit Dezimalpunkt oder –Komma formatiert sind. Entscheidend ist nur, dass kein Trennzeichen für die Tausendergruppen vorhanden ist, da sonst diese als Dezimaltrennung verwendet werden könnten!

Konnte der Text in **s** nicht in eine Zahl umgewandelt werden, liefert die Funktion **0** als Ergebnis.

3.1.4.3 Public Function PWDEncode() As String

Parameter: ByVal sOriginal As String

Diese Funktion dient dazu, Kennwörter und ähnliches zu verschlüsseln. Der Algorithmus ist recht einfach, zum Zweck der Verhüllung eines Kennwortes ist er jedoch ausreichend.

> **Achtung: Da diese Art der Verschlüsselung in fast jedem PMS-Programm identisch verwendet wird und außerdem einige Kennworte global in der Registrierung gespeichert werden, müssen alle Programme gleichzeitig geändert werden, sofern hier ein anderer Mechanismus eingesetzt werden soll!**

3.1.4.4 Public Function PWDDecode() As String

Parameter: ByVal sCrypt As String

Diese Funktion entschlüsselt einen mit **PWDEncode** verschlüsselten Text. Hier gelten die gleichen Warnungen wie bei **PWDEncode()**.

3.1.4.5 Public Function GetParameter() As Variant

Parameter: ByVal sParameter As String

Diese Funktion liest den Inhalt einer Variablen aus der Datenbanktabelle **[Int_Parameter]**. Die Variable wird durch den Parameter **sParameter** eindeutig definiert.

Wenn die angegebene Variable nicht gefunden wurde, wird kontrolliert, ob **sParameter** einen der folgenden Standard-Texte enthält. Wenn ja, wird als Ergebnis der Standardwert geliefert. Wenn nicht, wird einfach ein Fehler generiert.

Parameter	*Ergebnis*	*Beschreibung*
SAPR3_MANDANT	**g_sSAPMandant**	Der Mandantenstring für SAP. Der Standardwert ist hierbei "**450**".
SAPR3_TID	**g_sSAP_TID**	Die TID-Angabe für SAP. Der Standardwert ist hierbei " " (8 Leerzeichen).
SAPR3_WERK	**g_sSAPWerk**	Die Werksnummer für SAP. Der Standardwert ist hierbei "**4B**".
SAPR3_USER	**g_sSAPUser**	Der Benutzername für SAP. Der Standardwert ist hierbei "**SAP Transfer**".
STEPTWOINTERVAL	**20**	Das Intervall zwischen dem Initiieren einer Anforderung bei SAP und dem Auslesen des Ergebnisses. Der Standardwert ist hierbei "**20**".

Tabelle 9: Abfragen aus Parametertabelle

3.1.4.6 Public Function GetIpcCommand() As atTypes

Parameter: ByVal sParam As String

Diese Funktion prüft, ob in der PMS-DB (Tabelle **[IPC]**[8]) ein Kommando für den angegebenen Parameter **sParam**[9] gespeichert wurde (durch die Stammdatenverwaltung). Dabei werden nur die ersten 5 Zeichen der Kommandos ausgewertet und auch nur dann, wenn das Kommando nicht mit **„-OK“** endet. Wenn ja, liefert sie eines der folgenden Ergebnisse:

Eintrag	*Ergebnis*	*Beschreibung*
"AUFTR"	**atTypeOrderGet**	Es sollen Aufträge von SAP abgeholt werden.
"QMDAT"	**atTypeQmBatchGet**	Es sollen QM-Daten von SAP abgeholt werden.
"LIEFE"	**atTypeDeliverDataGet**	Es sollen Lieferdaten von SAP abgeholt werden.
"SPEZI"	**atTypeSpecGet**	Es sollen Spezifikationen von SAP abgeholt werden.
alles andere	**atTypeUnknown**	Es wird kein Sonderjob initialisiert

Tabelle 10:IPC-Kommandos

[8] IPC steht für *I*nter-*P*rocess-*C*ommunication, also eine Kommunikation zwischen verschiedenen Prozessen; in diesem Fall der Stammdatenverwaltung und dem SAP-Transfer.

[9] sParam enthält hier immer „InitTransfer“

3.2 basSQL

Dieses Modul enthält hauptsächlich konstant definierte Strings, die die notwendigen SQL-Befehle zur Kommunikation mit der Datenbank enthalten. Diese Strings sind meistens Vorlagen, die noch mit Parametern gefüllt werden müssen. Dazu dient die ebenfalls in diesem Modul liegende Funktion **`FormatSQL`**, die mit einer variablen Parameterliste gefüllt wird.

Weiterhin sind noch einige Funktionen und Prozeduren enthalten, die sich aber alle auf die Datenbank und ihre Behandlung beziehen.

3.2.1 Globale Konstanten

Alle hier vorliegenden Konstanten enthalten Strings zum Datenbankzugriff, die als **`Public`** und damit global definiert sind. Die teilweise angegebenen Parameter werden mit der Funktion **`FormatSQL()`** eingesetzt. Ist kein Parameter angegeben, so kann der String auch direkt verwendet werden.

3.2.1.1 sSqlConnect

Dies ist der Verbindungsstring zur Datenbank. Er wird gefüllt mit dem Servernamen, dem Benutzer und seinem Kennwort, sowie dem Namen der Datenbank (immer **`PMS`**).

3.2.1.2 sSqlP_CheckConnection

Dieses Statement dient nur zum Prüfen, ob die Datenbank ansprechbar ist. Es benötigt keine Tabelle und eigentlich auch keine Datenbank zur Ausführung.

3.2.1.3 sSqlP_GetAllErrors

Liefert die komplette Tabelle **`[Int_ErrorLog]`**. Dient zur Vorbereitung des Recordsets auf den Eintrag neuer Zeilen.

3.2.1.4 sSqlP_GetAllSrvLogs

Liefert die komplette Tabelle **`[Int_ServerLog]`**. Dient zur Vorbereitung des Recordsets auf den Eintrag neuer Zeilen.

3.2.1.5 sSqlP_GetParam

Parameter: *Parametername*
Parametergruppe

Liefert den Inhalt einer in **`[Int_Parameter]`** eingetragenen Variablen.

3.2.1.6 sSqlP_GetAuftrag

Parameter: Auftragsnummer

Liefert den Auftrag mit der gegebenen Nummer. Wird verwendet, wenn ein Auftrag von SAP übernommen werden soll, um zu entscheiden, ob ein bestehender Auftrag aktualisiert oder ein neuer eingefügt werden muss.

3.2.1.7 sSqlP_GetQmData

Parameter: Chargennummer
Merkmalname

Liefert die QM-Daten mit der gegebenen Chargennummer und Merkmal. Wird verwendet, wenn QM-Daten von SAP übernommen werden sollen, um zu entscheiden, ob bestehende Daten aktualisiert oder neue eingefügt werden müssen.

3.2.1.8 sSqlP_GetSpec

Parameter: Materialnummer
Verwendungsmerkmal

Liefert die Spezifikationen mit der gegebenen Materialnummer und Verwendungsmerkmal. Wird verwendet, wenn Spezifikationen von SAP übernommen werden sollen, um zu entscheiden, ob bestehende Daten aktualisiert oder neue eingefügt werden müssen.

3.2.1.9 sSqlP_GetLieferdaten

Parameter: Lieferantennummer
Chargennummer

Liefert die Lieferdaten mit der gegebenen Lieferanten- und Chargennummer. Wird verwendet, wenn Lieferdaten von SAP übernommen werden sollen, um zu entscheiden, ob bestehende Daten aktualisiert oder neue eingefügt werden müssen.

3.2.1.10 sSqlP_GetChargen

Liefert alle abgeschlossenen Chargen mit einem Transferstatus von **0** oder **A**, deren Anlagenteilstatus gesetzt ist (Diese Definition entspricht den in der Partieeröffnung abgeschlossenen Chargen). Hierbei werden alle Daten geliefert, die zum Übertragen ins SAP notwendig sind.

3.2.1.11 sSqlP_UpdChargenStat

Parameter: durch Komma getrennte Liste von Chargen

Dieses Kommando aktualisiert alle übergebenen Chargen, indem ihr Feld **[Status_Work]** von **0** auf **2**, bzw. von **A** auf **B**[10] gesetzt wird.

[10] A/B sind Abfallchargen

3.2.1.12 sSqlP_GetQmDaten

Holt alle Messwerte, deren Übertragungsstatus (**[Status_Work]**) **0** oder **A** enthält, also noch nicht versendet ist.

3.2.1.13 sSqlP_UpdQmDatenStat

Parameter: *durch Komma getrennte Liste von Kombinationen aus Chargennummer und Merkmalnamen*

Dieses Kommando aktualisiert alle durch Chargennummer und Merkmalnamen definierten Messwerte, indem ihr Feld **[Status_Work]** von **0** auf **2**, bzw. von **A** auf **B** gesetzt wird.

3.2.1.14 sSqlP_GetOpenOrders

Parameter: *Materialnummer*
System

Liefert Auftragsnummer und Startzeitpunkt der Aufträge, die auf dem angegebenen System zu der angegebenen Materialnummer passen und aktiv sind (Startzeitpunkt vorhanden und kleiner als das aktuelle Datum, kein Endezeitpunkt vorhanden).

3.2.1.15 sSqlP_InsTasklog

Parameter: *Jobtyp*
aktuelle Uhrzeit
Meldungstext
Jobstatus
Computername

Trägt die angegebenen Daten in die Tabelle **[Srv_AutoTaskLog]** ein.

3.2.1.16 sSqlP_GetTaskLog

Liefert alle Einträge aus der Sicht **[v_AutoTaskLog]**, die nicht älter als 3 Stunden sind. Wird zum Füllen der Liste verwendet.

3.2.1.17 sSqlP_GetFullTaskLog

Liefert alle Einträge aus der Sicht **[v_AutoTaskLog]**, die nicht älter als 30 Tage sind. Wird ebenfalls zum Füllen der Liste verwendet.

3.2.1.18 sSqlP_GetJob

Parameter: *Job-ID*

Lädt alle wichtigen Daten für den Job mit der angegebenen ID aus der Datenbanktabelle **[Srv_AutoTask]**.

3.2.1.19 sSqlP_GetAllJobs

Lädt alle wichtigen Daten für alle Jobs aus der Datenbanktabelle **[Srv_AutoTask]**.

3.2.1.20 sSqlP_UpdTask

Parameter: *Job-ID*
nächste Laufzeit
Anzahl Minuten zwischen den Aufrufen
Minimale Anzahl Sekunden zwischen 1. und 2. Teiljob
Aktivflag

Speichert den Zustand des Jobs in der Datenbank (Nur UPDATE; der Datensatz muss bereits vorhanden sein).

3.2.1.21 sSql_GetIPC

Parameter: *Gruppenname, hier immer „**InitTransfer**“*

Holt noch nicht erledigte Jobs für SAP-Transfer aus der Tabelle **[IPC]**. Es wird nur der Inhalt des Feldes **[Val]** zurückgeliefert, dies genügt jedoch, um zu sehen, ob und was zu tun ist.

3.2.1.22 sSql_ClearIPC

Parameter: *Gruppenname, hier immer „**InitTransfer**“*
Wert (Feld [Val])

Setzt alle Einträge der Tabelle [IPC] auf erledigt, die zur Gruppe „Transfer“ gehören und den angegebenen Wert enthalten.

3.2.2 Funktionen

3.2.2.1 Public Function FormatSQL() As String

Parameter: *ByVal sFormat As String*
ParamArray par() As Variant

Da Visual Basic — anders als andere Programmiersprachen — keine hochflexible Formatierung unterstützt, musste zum Formatiere der SQL-Kommandos diese Funktion geschaffen werden.

Der erste Parameter — **sFormat** — enthält dabei nummerierte Platzhalter im Format **%1**...**%n**, die durch die weiteren Parameter — in **par()** — ersetzt werden. Dabei wird jedes Auftreten von **%1** durch den ersten, jedes **%2** durch den zweiten Parameter, usw. ersetzt.

Die Parameter werden dabei zuerst in Strings konvertiert (mit **CStr()**).

Wenn weniger Parameter angegeben werden, als Platzhalter vorhanden sind, so verbleiben die überzähligen Platzhalter im Ergebnisstring.

```
Dim sSql As String, sFormat As String
sFormat = "SELECT [Val] FROM [IPC] WHERE [Betrieb]='%1' AND [Param]='%2'"
sSql = FormatSQL(sFormat, "P2K", "InitTransfer")

Danach enthält sSql den fertig formatierten String

"SELECT [Val] FROM [IPC] WHERE [Betrieb]='P2K' AND [Param]='InitTransfer'"
```

Code 2: Beispiel zu FormatSQL

3.2.2.2 Public Function CheckPMS() As Boolean

Parameter: Keine

Diese Funktion prüft, ob die Datenbank erreichbar ist. Hierbei wird *nicht* versucht, eine fehlende Verbindung aufzubauen!

Ist die Datenbank mit den aktuellen Einstellungen ansprechbar, so liefert die Funktion **True** zurück, sonst **False**.

3.2.2.3 Public Function ReconnectPMS() As Boolean

Parameter: Keine

Diese Funktion stellt eine Verbindung zur Datenbank her. Als erstes wird dabei eine eventuell bereits bestehende Verbindung zur Datenbank geschlossen. Danach werden die Verbindungsparameter *auf jeden Fall* wieder aus der Registrierung gelesen. Durch diesen einfachen Mechanismus lassen sich auch remote initiierte Umstellungen der Datenbank, etc. einfach realisieren.

Wurde die Verbindung hergestellt, so liefert die Funktion das Ergebnis **True** und das Verbindungsobjekt **g_PMSConn** ist fertig initialisiert.

Trat ein Fehler auf, wird **False** geliefert und **g_PMSConn** enthält **Nothing**.

3.3 basSAP

Auch dieses Modul wird — wie **basSQL** — hauptsächlich für konstante Strings verwendet. Der Unterschied besteht darin, dass die hier abgelegten Daten speziell für den Zugriff auf die Oracle-Datenbank des SAP-Systems gedacht sind.

3.3.1 Globale Konstanten

Alle hier vorliegenden Konstanten enthalten Strings zum Zugriff auf SAP, die als **Public** und damit global definiert sind. Die Parameter werden mit der Funktion **SAPFormat()** eingesetzt (Im Gegensatz zu den Konstanten in **basSQL** benötigen hier alle Strings Parameter).

3.3.1.1 sSAPConnect

Parameter: *TNS-Name*
Benutzername
Kennwort

Dies ist der Verbindungsstring zur Oracle-Datenbank. Er wird gefüllt mit dem TNS-Namen, dem Benutzernamen und dem zugehörigen Kennwort.

3.3.1.2 sSAP_CheckConnection

Parameter: *Werksnummer*

Dient zum Überprüfen der Verbindung zum SAP. Das Ergebnis (die Anzahl der Datensätze in der Aktionstabelle) ist hier uninteressant, es ist nur wichtig, ob das Kommando ausgeführt werden kann oder nicht.

3.3.1.3 sSAP_InsMesswert

Parameter: *Werksnummer*
Mandant
Materialnummer
Chargennummer
Verwendungsmerkmal
Messwert
Grenzwert Max
Grenzwert Min
TID
Datum
Uhrzeit
Status

Sendet den Messwert mit allen benötigten Daten an die SAP-Tabelle **[ZMESSWERT_VP_4B**[11]**]**.

[11] **4B** ist das Werkskürzel für KoSa Offenbach, das hier immer ein Teil des Tabellennamens ist, teilweise aber auch noch in den Tabellen angegeben werden muss.

3.3.1.4 sSAP_UpdMesswert

Parameter: *Werksnummer*
Messwert
Chargennummer
Verwendungsmerkmal

Aktualisiert den Messwert in der Tabelle **[ZMESSWERT_VP_4B]**, falls er dort schon vorhanden ist. Wird als Notfallmechanismus beim Senden der Messwerte verwendet.

3.3.1.5 sSAP_InsCharge

Parameter: *Werksnummer*
Mandant
Auftragsnummer
Materialnummer
Chargennummer
Chargengewicht
Startdatum
Startzeit
Endedatum
Endezeit
Lagerplatz
TID
Datum
Uhrzeit
Status

Sendet die Charge mit allen benötigten Daten an die SAP-Tabelle **[ZCHARGE_VP_4B]**.

3.3.1.6 sSAP_UpdCharge

Parameter: *Werksnummer*
Chargengewicht
Endedatum
Endezeit
Chargennummer

Aktualisiert die Charge in der Tabelle **[ZCHARGE_VP_4B]**, falls er beim Senden dort bereits vorhanden ist. Wird als Notfallmechanismus beim Senden der Chargen verwendet.

3.3.1.7 sSAP_InsAktion

Parameter: *Werksnummer*
Mandant
TID
Erfassungsdatum
Erfassungszeit
Aktionstyp

Schreibt einen Eintrag in die Aktionstabelle **[ZAKTION_VP_4B]**. Je nach Aktionstyp (siehe dazu **SAPAktionConstants** in **CSapGlobals** auf Seite ???) ist das eine Aufforderung an SAP, Daten zu senden oder die Benachrichtigung, dass in einer anderen Tabelle Daten zur Abholung bereit stehen.

3.3.1.8 sSAP_GetOrder

Parameter: *Werksnummer*
Mandant

Holt alle Daten aus der Tabelle **[ZAUF_VP_4B]** mit gültiger Auftragsnummer und Status 4 (bereitgestellte, aber noch nicht abgeholte Daten).

3.3.1.9 sSAP_UpdOrder

Parameter: *Werksnummer*
Mandant

Stellt den Sendestatus der Auftragsdaten mit gültiger Auftragsnummer auf 6 (Daten wurden abgeholt).

3.3.1.10 sSAP_GetQmBatch

Parameter: *Werksnummer*
Mandant

Holt alle Daten aus der Tabelle **[ZQM_BATCH_VP_4B]** mit gültiger Chargennummer oder gültigem Merkmalnamen und Status 4 (bereitgestellte, aber noch nicht abgeholte Daten).

3.3.1.11 sSAP_UpdQmBatchStat

Parameter: *Werksnummer*
Mandant

Stellt den Sendestatus der QM-Daten mit gültiger Chargennummer oder gültigem Merkmalnamen auf 6 (Daten wurden abgeholt).

3.3.1.12 sSAP_GetSpec

Parameter: *Werksnummer*
Mandant

Holt alle Daten aus der Tabelle **[ZSPEZ_VP_4B]** mit gültiger Materialnummer oder gültigem Verwendungsmerkmal und Status 4 (bereitgestellte, aber noch nicht abgeholte Daten).

3.3.1.13 sSAP_UpdSpecStat

Parameter: *Werksnummer*
Mandant

Stellt den Sendestatus der Spezifikationen mit gültiger Materialnummer oder gültigem Verwendungsmerkmal auf 6 (Daten wurden abgeholt).

3.3.1.14 sSAP_GetLieferdaten

Parameter: *Werksnummer*
Mandant

Holt alle Daten aus der Tabelle **[ZQM_LIEFER_VP_4B]** mit gültiger Lieferantennummer oder gültiger Chargennummer und Status 4 (bereitgestellte, aber noch nicht abgeholte Daten).

3.3.1.15 sSAP_UpdLieferdatenStat

Parameter: *Werksnummer*
Mandant

Stellt den Sendestatus der Lieferdaten mit gültiger Lieferantennummer oder gültiger Chargennummer auf 6 (Daten wurden abgeholt).

3.3.2 Funktionen

3.3.2.1 Public Function SAPFormat() As String

Parameter: *ByVal sFormat As String*
ParamArray par() As Variant

Diese Funktion dient — genau wie **FormatSQL()**, siehe dazu Seite 15 — zum Ausfüllen von SQL-Kommandos mit variablen Parametern. Der einzige Unterschied besteht darin, dass diese Funktion (**SAPFormat()**) darauf achtet, dass Leerstrings in den Parametern durch ein einzelnes Leerzeichen ersetzt werden, da Oracle, das für die SAP-Datenbank verwendet wird, ansonsten einen Fehler produziert.

3.3.2.2 Public Function CheckSAP() As Boolean

Parameter: *Keine*

Diese Funktion kontrolliert, ob das Verbindungsobjekt zur Oracle-Datenbank vorhanden und geöffnet ist. Wenn ja, wird weiterhin getestet, ob ein einfaches Kommando abgesetzt werden kann. Funktioniert das alles, liefert die Funktion **True**. Schlägt einer dieser Tests fehl, so kommt sie mit **False** zurück.

3.3.2.3 Public Function ReconnectSAP() As Boolean

Parameter: Keine

Diese Funktion stellt eine Verbindung zur SAP-Datenbank her. Als erstes wird dabei eine eventuell bereits bestehende Verbindung zur Datenbank geschlossen und zerstört. Danach werden die Verbindungsparameter *auf jeden Fall* wieder aus der Registrierung gelesen und versucht, mit diesen Angaben eine neue Verbindung herzustellen. Durch diesen einfachen Mechanismus lassen sich auch remote initiierte Umstellungen der Datenbank, etc. einfach realisieren.

Wurde die Verbindung hergestellt, so liefert die Funktion das Ergebnis **`True`** und das Verbindungsobjekt **`g_SAPConn`** ist fertig initialisiert. Außerdem wird der Status der Überwachungsklasse **`CSystemStatus`** auch auf **`True`** gesetzt.

Trat ein Fehler auf, wird **`False`** geliefert und **`g_SAPConn`** enthält **`Nothing`**.

3.4 MultiHook

Dieses Modul dient dazu, Standard Windows-Fensternachrichten zu verarbeiten. Im hier beschriebenen **PMS_Transfer** wird es ausschließlich dazu verwendet, die Windows-Nachricht **WM_GETMINMAXINFO** sinnvoll zu verarbeiten[12].

Sehr wichtig hierbei ist, dass die Funktionalität nur im fertigen Programm greift; während dem Debuggen wäre es verheerend, eine zweite Nachrichtenbehandlung zu aktivieren — VB bleibt dann einfach stehen!

3.4.1 Datentypen und Enums

```
Public Type POINTAPI
   x     As Long
   y     As Long
End Type
```

Tabelle 11: Type POINTAPI

Dieser Typ wird in der Struktur **MINMAXINFO** verwendet.

```
Public Type MINMAXINFO
   ptReserved      As POINTAPI
   ptMaxSize       As POINTAPI
   ptMaxPosition   As POINTAPI
   ptMinTrackSize  As POINTAPI
   ptMaxTrackSize  As POINTAPI
End Type
```

Tabelle 12: Type MINMAXINFO

Die Struktur **MINMAXINFO** ist ein Abbild der Originalstruktur aus dem Windows-System und wird dazu verwendet, in der Behandlung der Nachricht **WM_GETMINMAXINFO** die Minimalgröße des Fensters festzulegen.

3.4.2 Konstanten

GWL_WNDPROC (**-4**) ist ein Standardwert, der auf den Bereich in der allgemeinen Fensterstruktur zeigt, in dem der Zeiger auf die Fensterfunktion dieses Fensters abgelegt ist.

WM_GETMINMAXINFO (**&H0024**) ist eine Windows-Nachricht, die bei jeder Größenänderung an das Fenster gesendet wird. Dabei wird als Parameter ein Zeiger auf eine Struktur vom Typ **MINMAXINFO** mitgeliefert.

3.4.3 Deklarierte Funktionen

Die hier deklarierten Funktionen sind Funktionen aus dem normalen Windows-Bereich. Sie werden behandelt wie normale VB-Funktionen. Eine genauere Beschreibung findet sich in der Hilfe zur Windows-API.

[12] Es ist traurig, dass so etwas überhaupt notwendig ist, aber VB unterstützt nur einen begrenzten Pool an Standard-Nachrichten!

3.4.3.1 GetModuleFileName

Liefert den Dateinamen des gerade ausgeführten Moduls.

3.4.3.2 CopyMemory

Kopiert einen beliebigen Speicherbereich in einen anderen. Wird bei der Behandlung der Windows-Nachricht **WM_GETMINMAXINFO** verwendet.

3.4.3.3 CallWindowProc

Ruft die Fensterfunktion eines Fensters auf.

3.4.3.4 SetWindowLong

Setzt einen fenster-spezifischen Long-Wert. In dieser Klasse wird damit die Fensterfunktion für ein Fenster festgelegt.

3.4.4 Lokale Variablen

3.4.4.1 m_HookedForms As Collection

Kollektion, die Zeiger auf alle Fenster (Formen) enthält, die diesen Mechanismus verwenden.

3.4.4.2 m_OldWndProcs As Collection

Kollektion, die für alle behandelten Fenster die Zeiger auf die Original-Fensterfunktionen enthält.

3.4.5 Öffentliche Prozeduren

3.4.5.1 Public Sub HookForm()

Parameter: frm As Form

Mit dieser Methode wird im übergebenen Fenster die erweiterte Behandlung der Nachrichten aktiviert.

Solange das Programm unter der IDE[13] läuft, bewirkt diese Methode gar nichts, um die verschiedenen Probleme zu vermeiden, die sich unter der IDE sonst ergäben.

Im realen Modus wird die bisherige Fensterprozedur gespeichert und **newWndProc** als neue dem Fenster zugewiesen.

[13] IDE = Integrated Development Environment, die Entwicklungs-Oberfläche.

3.4.5.2 Public Sub UnhookForm()

Parameter: *frm As Form*

Diese Prozedur ist das Gegenstück zu **HookForm()**: Sie nimmt alle vorgenommenen Änderungen zurück.

Es ist entscheidend, dass diese Methode vor dem Beenden des Programms, bzw. vor dem Schließen des Fensters, aufgerufen wird. Ansonsten wäre ein Programmabsturz sicher, da interne Systemzeiger auf Funktionen zeigen würden, die nicht mehr im Speicher liegen.

Unter der IDE hat diese Methode natürlich ebenfalls keine Auswirkungen.

3.4.5.3 Public Sub UnhookAllForms()

Parameter: *Keine*

Diese Methode führt **UnhookForm()** für alle überwachten Fenster durch.

3.4.5.4 Public Sub GetMinMax()

Parameter: *ByVal MinMax As Long*
MinX As Long
MinY As Long
Optional MaxX As Long
Optional MaxY As Long

Diese Prozedur kann als Reaktion auf die Nachricht **WM_GETMINMAX** aufgerufen werden, um dem System die gewünschten Werte mitzuteilen.

Achtung: Diese Prozedur verwendet Pixel (also reale Bildschirmpunkte) für die Größenangaben! In VB wird üblicherweise mit der Maßeinheit TWIP[14] gearbeitet, die deutlich kleiner sind! Daher wurde auch die nächste Methode eingeführt:

3.4.5.5 Public Sub GetMinMaxTwips()

Parameter: *ByVal MinMax As Long*
MinX As Long
MinY As Long
Optional MaxX As Long
Optional MaxY As Long

Diese Methode führt die gleichen Funktionen aus wie **GetMinMax()**, jedoch werden hier die Parameter in Twips erwartet und innerhalb der Methode umgerechnet. Ein Beispiel aus dem aktuellen Programm:

```
' die Standardgröße des Fensters ist das Minimum
m_lMinWidth = Width
m_lMinHeight = Height

HookForm Me ' Subclassing für GetMinMaxInfo aktivieren
```

Code 3: frmMain / Form_Load

[14] Ein TWIP ist eine Maßeinheit aus dem industriellen Drucken. Sie bedeutet „Twentieth of a Point“ und umfasst $^1/_{1440}$ Zoll. In VB wird sie üblicherweise als Dezimalzahl dargestellt.

Hier werden zwei Modulvariablen mit der Fenstergröße initialisiert und das Modul initialisiert.

```
Public Function Form_Message(uMsg As Long, wParam As Long, lParam As Long) _
        As Boolean
   Form_Message = False
   Select Case uMsg
      Case WM_GETMINMAXINFO
         GetMinMaxTwips lParam, m_lMinWidth, m_lMinHeight
         Form_Message = True  ' Message wurde abgearbeitet
   End Select
End Function
```

Code 4: frmMain / Form_Message

In dieser Methode, die von **MultiHook** direkt aufgerufen wird (siehe **newWndProc()**, Seite 25), wird — nur bei der Nachricht **WM_GETMINMAXINFO** — die Prozedur **GetMinMaxTwips()** aufgerufen, die die Parameter vorbereitet und in die Struktur, auf die der Zeiger in **lParam** zeigt, geschrieben. Auf diese Weise wird die Systemanfrage den Regeln von Windows entsprechend bearbeitet.

3.4.6 Öffentliche Funktionen

3.4.6.1 Public Function IsInIDE() As Boolean

Parameter: *Keine*

Diese Funktion liefert **True**, wenn das Programm unter der IDE läuft, sonst **False**. Der Test geschieht über die Abfrage des Dateinamens des aktuellen Moduls. Unter der DIE enthält dieser auf jeden Fall den String **„VB6.EXE“**[15].

Es muss daher darauf geachtet werden, dass der Name des erstellten Programms den Text „VB6.EXE“ nicht enthält, da sonst das Modul niemals greifen würde!

3.4.7 Interne Funktionen

3.4.7.1 Private Function newWndProc() As Long

Parameter: *ByVal hWnd As Long*
ByVal uMsg As Long
ByVal wParam As Long
ByVal lParam As Long

Diese Funktion wird in **HookForm()** als neue Fensterfunktion festgelegt und danach vom System aufgerufen. Sie sollte auf keinen Fall vom Programm her direkt aufgerufen werden!

Zuerst wird geprüft, ob das aufrufende Fenster (das Handle wird in **hWnd** geliefert) überhaupt in der Liste eingetragen ist. Wenn nicht, wird die Funktion sofort abgebrochen.

[15] Das funktioniert natürlich nur unter der Version 6 von Visual Basic. Da die nächste Version (VB.NET) aber ein völlig anderes Konzept verwendet und eine Portierung nur unter Schwierigkeiten möglich ist, stört diese Einschränkung hier nicht weiter.

Sonst wird die Fenster-interne Funktion **Form_Message()** (siehe **frmMain** ab Seite 27, sowie das Beispiel bei **GetMinMaxTwips()** auf Seite 24) mit den Parametern **uMsg**, **wParam** und **lParam** aufgerufen, die dann wiederum in beliebiger Weise mit den Informationen verfahren kann. Liefert **Form_Message() True** als Ergebnis, so wird die Bearbeitung beendet, ansonsten wird die Original-Fensterfunktion aufgerufen.

Es ist natürlich auch möglich, sowohl die neue Funktion **Form_Message()**, als auch die Standard-Fensterfunktion aufzurufen, indem **Form_Message()** einfach **False** als Ergebnis liefert.

ACHTUNG: Wenn ein Fenster in dieses Modul eingebunden wird, so muss es die Funktion Form_Message() enthalten!

4 Fenster

Unter dem Begriff „Fenster“ werden alle so genannten Formen, Dialoge und ähnliches zusammengefasst.

4.1 frmMain

Dieses Fenster (eigentlich ist es ein Dialog, es wird jedoch wie ein Fenster angezeigt) bietet die Hauptoberfläche des Programms. Hier können das System des aktuellen Betriebes und eine der letzten 10 Chargen ausgewählt werden, um Informationen dazu anzuzeigen.

4.1.1 Bildschirm-Elemente

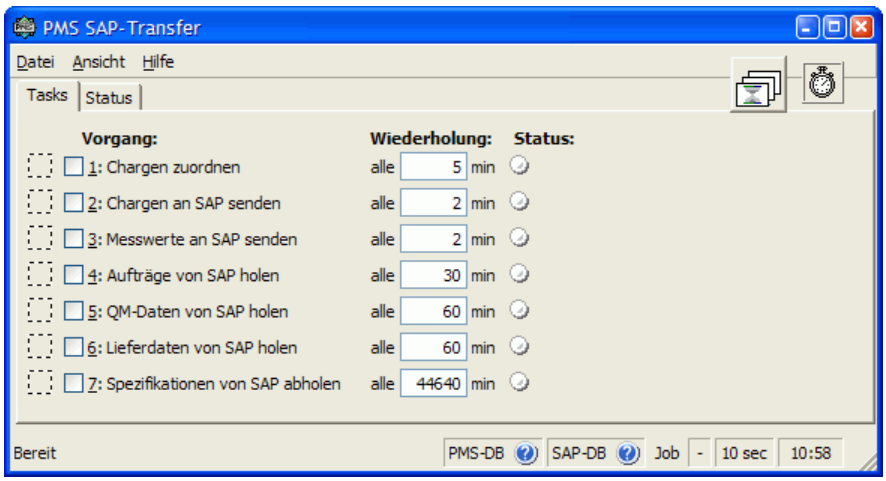

Abbildung 1: frmMain

Elemente im Fenster:

- Menüleiste. Das übliche Hauptmenü des Programms.
- Tabulatorleiste, mit der zwischen der Anzeige der Tasks und der Statusliste umgeschaltet werden kann.
- Sieben Bildelemente zur Anzeige des jeweils aktiven Tasks.
- Sieben Checkboxen zum Aktivieren und Deaktivieren der Tasks.
- Laufende Nummern und Bezeichnung der Tasks.
- Eingabefelder zur Einstellung der Wiederholungsrate der Tasks.
- Statussymbole zum einfachen Erkennen des Status der Tasks.
- Kommentarfelder, die mit dem jeweils letzten Ergebnis der Tasks gefüllt werden.
- Ein Timer, der das Programm zum Laufen bringt.
- Image-Liste, die einige Symbole für die Aktiv-Bildelemente, die Statussymbole, die Statusleiste sowie die Symbole für die Anzeige im Traybereich enthält.
- Ein Listenelement (hier nicht sichtbar), in dem die Statusmeldungen angezeigt werden.

- Statusleiste. In dieser werden folgende Infos dargestellt:
  Statusbereich: Zur Anzeige des aktuellen Programmstatus.
  PMS-DB: Der Name und der Status der Verbindung zur PMS-DB.
  SAP-*DB:* Der Name und der Status der Verbindung zu SAP.
  Job: Die Nummer des zuletzt ausgeführten Jobs.
  Wartezeit: Die Zeit, die zwischen zwei Jobs gewartet wird.
  Uhrzeit: Die aktuelle Uhrzeit.

4.1.2 Lokale Variablen

4.1.2.1 Private WithEvents m_SysTray As CSysTray

Diese Klasse unterstützt das Ablegen des Programms im Systemtray.

4.1.2.2 Private m_Job(0 To 6) As CJob

Ein Array mit sieben Elementen der Klasse **CJob**, die prinzipiell die eigentliche Arbeit übernehmen.

4.1.2.3 Private m_bKilling As Boolean

Dieser Schalter steht im Normalfall auf **False** und wird auf **True** gestellt, wenn das Programm beendet werden soll. Dadurch kann eine doppelte Nachfrage beim Beenden vermieden werden (In der Sub **Form_QueryUnload**).

4.1.2.4 Private m_nCurrentJob As Integer

Index, der den aktuellen Job bezeichnet.

4.1.2.5 Private m_bInitPhase As Boolean

Schalter, der zu Beginn auf **True** und nach dem Abarbeiten der Initialisierungsphase auf **False** gesetzt wird. Er wird in einigen Prozeduren und Funktionen abgefragt, um zu vermeiden, dass Ereignisse ausgelöst werden, zu denen das Programm noch nicht bereit ist.

4.1.2.6 Private m_tmNextCheck As Date

Diese Datumsvariable enthält den Zeitpunkt, zu dem der nächste Test durchgeführt werden soll. Da der Timer auf sekündliche Intervalle eingestellt ist, um das System etwas reaktiver zu halten, wird zu Beginn der Timerroutine einfach verglichen, ob die aktuelle Uhrzeit kleiner als die Zeit in **m_tmNextCheck** ist. Wenn ja, wird die Prozedur einfach wieder verlassen.

4.1.2.7 Private m_lMinWidth, m_lMinHeight As Long

In diesen beiden Variablen wird zu Beginn der Prozedur **Form_Load()** die Größe des Dialogs gespeichert. In der Funktion **Form_Message()** werden die beiden dann als minimale Größe des Fensters verwendet.

4.1.3 Interne Prozeduren

4.1.3.1 Private Sub Form_Load()

Parameter: Keine

Diese Prozedur wird vom System aufgerufen, sobald das Fenster geladen wird[16].

Als erstes wird das Objekt **m_SysTray** initialisiert, mit dem das grüne Symbol T im Systray gesteuert wird.

Danach wird mit der Methode **HookForm()** das Modul **MultiHook** aktiviert, das dafür sorgt, dass eine bestimmte Mindestgröße nicht unterschritten wird.

Nun werden die zuletzt gespeicherten Positionsinformationen aus der Registrierung geladen und angewandt.

Jetzt werden die 7 Jobs als Objekte der Klasse **CJob** initialisiert und zwar jeweils einer für eine bestimmte Tätigkeit:

1. Chargen zuweisen.
2. Chargen senden.
3. Messwerte senden.
4. Aufträge anfordern / abholen.
5. QM-Daten anfordern / abholen.
6. Lieferdaten anfordern / abholen.
7. Spezifikationen anfordern / abholen.

Aus diesen so erstellten Jobs werden dann auch Informationen wie der Titel, der Status, etc. geholt und im Fenster angezeigt.

Abschließend wird noch die Statusleiste initialisiert, der Timer aktiviert und auf 5 Sekunden gestellt, um den ersten Aufruf schnell zu erhalten.

4.1.3.2 Private Sub Form_Resize()

Parameter: Keine

Diese Prozedur wird vom System aufgerufen, sobald sich die Fenstergröße ändert. Ist das Fenster minimiert, so sorgt die Methode **m_SysTray.MinToSysTray()** dafür, dass der normale Eintrag in der Windows-Statusleiste verschwindet und das Symbol im Systray dargestellt wird.

Wird das Fenster dagegen normal dargestellt, so werden die Positionen und Größen der einzelnen Fensterelemente angepasst.

[16] Geladen wird das Fenster, sobald darauf zugegriffen wird. In diesem Fall geschieht das in der Prozedur **Main()** im Modul **basMain**.

4.1.3.3 Private Sub Form_Unload()

Parameter: Cancel As Integer

Diese Prozedur wird beim Entladen des Fensters — und zwar direkt vor dem endgültigen Schließen — vom System aufgerufen. Hier wird die Kontrolle durch **MultiHook** wieder deaktiviert. Dann wird das Symbol aus dem Systray entfernt und Position und Größe des Fensters in der Registrierung gespeichert. Als nächstes wird ein Hinweis darauf, dass das Programm beendet wurde, im Eventlog und in der Datenbank eingetragen.

Dann wird die Status-Überwachung dieses Programms beendet und die Verbindung zur Datenbank geschlossen.

Danach kann das Fenster problemlos geschlossen und vernichtet werden.

4.1.3.4 Private Sub Form_QueryUnload()

Parameter: Cancel As Integer
UnloadMode As Integer

Diese Methode wird vom System aufgerufen, wenn das Fenster geschlossen werden soll, und zwar noch vor **Form_Unload()**. Je nachdem, aus welchem Grund dieses Schließen geschehen soll, wird hier unterschiedlich reagiert:

- Wenn Windows heruntergefahren wird oder das Programm über den Taskmanager beendet wird, wird das einfach so akzeptiert.
- Wenn das Programm über das Systemmenü oder das **x** in der Titelleiste geschlossen werden soll, wird der Parameter **Cancel** auf True gesetzt, wodurch das Beenden abgebrochen wird. Außerdem wird das Fenster in den Systray minimiert.
- Bei jedem anderen Grund (z.B. dem Menüeintrag „Beenden“) wird der Anwender gefragt, ob er das auch ernst meint. Antwortet er *nicht* mit Ja, so wird auch jetzt wieder das Beenden abgebrochen und das Programm läuft normal weiter.
  An dieser Stelle gilt noch die Ausnahme, dass, wenn das Programm unter der IDE läuft, die Nachfrage unterdrückt und das Programm einfach beendet wird.

4.1.3.5 Private Sub Form_KeyDown()

Parameter: KeyCode As Integer
Shift As Integer

Diese Prozedur kontrolliert, ob die Tastenkombination **Strg**-**Tab** oder **Umschalt**-**Strg**-**Tab** getippt wurde. Wenn ja, wird die Tabulatorleiste **m_Tabs** umgeschaltet, wodurch dann zwischen der normalen Anzeige und der Statusliste gewechselt wird.

4.1.3.6 Private Sub m_Bar_ToolClick()

Parameter: ByVal Tool As ActiveBar2LibraryCtl.Tool

Wird aufgerufen, sobald einer der Menüpunkte angeklickt wurde. Je nach Menüpunkt wird dann die entsprechende Routine aufgerufen, bzw. auch direkt reagiert.

4.1.3.7 Private Sub m_State_ColumnClick()

Parameter: ByVal ColumnHeader As MSComctlLib.ColumnHeader

Wenn der Benutzer eine der Kopfzeilen der Statusliste anklickt, wird die Sortierung in der Liste entsprechend angepasst. Wie auch im normalen Windows Explorer wechselt die Sortierordnung beim zweiten Klick auf die gleiche Spalte zwischen auf- und absteigend.

4.1.3.8 Private Sub m_Timer_Timer()

Parameter: Keine

Diese Prozedur wird immer dann aufgerufen, wenn der Timer abgelaufen ist. Als erstes wird dann kontrolliert, ob die Uhrzeit für die nächste Prüfung schon erreicht ist. Wenn nicht, wird die Prozedur sofort wieder verlassen.

Sonst wird als erstes der Timer deaktiviert, um Überschneidungen zu verhindern, falls der auszuführende Job länger benötigt.

Danach wird die Funktion **`CheckJobs()`** (siehe Seite 35) ausgeführt, die die ganze joborientierte Arbeit übernimmt.

Zum Abschluss wird die Wartezeit zwischen den Jobs aus der Registrierung gelesen, zur aktuellen Zeit addiert und dieser Wert als Startzeit für den nächsten Job gespeichert. Als letztes wird der Timer wieder aktiviert.

4.1.3.9 Private Sub m_imgStatus_DblClick()

Parameter: Index As Integer

Wird aufgerufen, wenn der Benutzer einen Doppelklick auf einer der Statusanzeigen zwischen Beschreibung und Status der Jobs ausführt. In diesem Fall wird — sofern der Job nicht deaktiviert ist — seine nächste Laufzeit auf „Jetzt“ gesetzt. Das hat den Effekt, dass der Job ausgeführt wird, sobald die normale Schleife bei seinem Index angekommen ist.

4.1.3.10 Private Sub SetJobState()

Parameter: ByVal i As Integer
Optional ByVal bLogIt As Boolean = True

Diese Methode zeigt den Status des Jobs mit dem Index **`i`** im Fenster an. Dabei wird die Farbe des „Statuslämpchens“, sowie der Text im Statusfeld eingestellt.

Farbe	*Beschreibung*
Weiß	Status ist in Ordnung.
Gelb	Job wartet auf Antwort von SAP, bzw. hat noch keine Daten zum Senden.
Rot	Job ist fehlgeschlagen.
Orange	Job ist fehlgeschlagen, wartet aber noch auf Antwort.
Blau	Job ist geplant, die Startzeit wird angezeigt.
Grün	Job ist zurzeit gerade aktiv.
Grau	Job ist deaktiviert.

Tabelle 13: Farben der Statusanzeige

Ist der Parameter **`bLogIt True`**, so wird der Jobstatus in der Datenbank geloggt.

Außerdem wird die Anzeige im Systemtray noch eingestellt: Wenn der Job fehlgeschlagen ist, wird das Traysymbol rot (T) angezeigt, sonst grün (T).

4.1.3.11 Private Sub AddLogError()

Parameter: *ByVal s1 As String*
ByVal s2 As String

Fügt den durch die Parameter definierten Text als Fehler in die Statusliste ein.

4.1.3.12 Private Sub FillLog()

Parameter: *ByVal bFull As Boolean*

Diese Methode füllt die Statusliste mit den Logeinträgen, die in der Datenbank gespeichert sind. Ist **`bFull True`**, so werden die Einträge der letzten 30 Tage geholt. Ansonsten werden nur die letzten 8 Stunden eingesetzt.

Dabei wird die Liste zuerst geleert, um keine doppelten Daten anzuzeigen.

4.1.3.13 Private Sub m_chkJob_Click()

Parameter: *Index As Integer*

Wird aufgerufen, wenn eine der 7 Checkboxen umgeschaltet wurde. Je nach neuer Eigenschaft wird dadurch der Job aktiviert oder deaktiviert.

4.1.3.14 Private Sub m_SysTray_LButtonDown()

Parameter: *Keine*

Wird aufgerufen, wenn der Benutzer mit der linken Maustaste auf das Symbol im Traybereich klickt. Je nach Programmstatus (sichtbar / unsichtbar) wird das Fenster entweder versteckt oder wieder sichtbar gemacht.

4.1.3.15 Private Sub m_SysTray_RButtonUp()

Parameter: *Keine*

Öffnet das Kontextmenü, wenn der Benutzer mit der rechten Maustaste auf das Symbol im Traybereich klickt.

4.1.3.16 Private Sub m_Tabs_Click()

Parameter: *Keine*

Methode wird aufgerufen, wenn der Benutzer einen Reiter der Tabulatorleiste auswählt. Dies geschieht auch dann, wenn diese Umschaltung durch die Tastatur erzeugt wurde (siehe dazu auch **`Form_KeyDown()`** auf Seite 30). Je nach Index des Controls werden dann unterschiedliche Elemente sichtbar geschaltet und neu gezeichnet, um das Umschalten möglichst natürlich wirken zu lassen.

4.1.3.17 Private Sub m_txtRepeat_GotFocus()

Parameter: *Index As Integer*

Wird aufgerufen, wenn eine der sieben Textboxen für die Wiederholrate den Focus erhält. Hier wird dann (so, wie in normalen Windows-Programmen üblich) der komplette Inhalt des Feldes markiert.

4.1.3.18 Private Sub m_txtRepeat_LostFocus()

Parameter: *Index As Integer*

Wird aufgerufen, wenn eine der Textboxen für die Wiederholrate verlassen wird. Der Inhalt des Textfeldes wird gelesen und — sofern der Inhalt nicht **0** ist — dem entsprechenden Job als Wartezeit zugewiesen.

4.1.3.19 Private Sub m_txtRepeat_KeyPress()

Parameter: *Index As Integer*
KeyAscii As Integer

Wird aufgerufen, wenn in einer der Textboxen für die Wiederholrate ein Zeichen eingegeben wird. Alle Zeichen außer den Ziffern und Backspace (ç) werden zurückgewiesen, da hier nur Ganzzahlen erlaubt sind.

4.1.3.20 Private Sub AllEntries()

Parameter: *Keine*

Wird aufgerufen, wenn das Symbol oder der Menüeintrag für „Alle Einträge“ angeklickt wird. Dabei wird **`FillLog()`** mit dem neuem Status des Umschalters aufgerufen.

4.1.3.21 Private Sub mn_Exit_Click()

Parameter: *Keine*

Wird aufgerufen, wenn der Menüpunkt „Beenden“ aufgerufen wird. Hier wird einfach versucht, das Fenster zu entladen. Ob das tatsächlich zum Beenden des Programms führt, wird in der dadurch aufgerufenen Routine **`Form_QueryUnload()`** entschieden.

4.1.3.22 Private Sub ReloginPMS()

Parameter: *Keine*

In dieser Prozedur wird zuerst der Dialog zur Eingabe der Anmeldedaten angezeigt. Wurde dieser mit **OK** bestätigt, so werden nun die gespeicherten Informationen aus der Registrierung gelesen und dazu verwendet, die Verbindung zur PMS-Datenbank herzustellen.

Hierzu wird das Verbindungsobjekt ggf. geschlossen und auf jeden Fall zerstört. Danach wird es mit den gelesenen Angaben neu erstellt und der Name des Servers in der Statusleiste angezeigt.

ACHTUNG: Schlägt diese Routine fehl, empfiehlt es sich, das Programm zu beenden und neu zu starten, da ohne DB-Verbindung eine weitere Arbeit unmöglich ist!

4.1.3.23 Private Sub ReloginSAP()

Parameter: Keine

Diese Prozedur macht das Gleiche wie **`ReloginPMS()`**, mit dem Unterschied, dass hier die SAP-Datenbank verbunden wird.

4.1.3.24 Private Sub mn_TrayExit_Click()

Parameter: Keine

Wird aus dem Kontextmenü des Traysymbols aufgerufen. Gibt die Kontrolle einfach an die zuvor erwähnte Prozedur **`mn_Exit_Click()`** weiter.

4.1.3.25 Private Sub mn_Hide_Click()

Parameter: Keine

Wird aus dem Kontextmenü des Traysymbols aufgerufen, wenn das Programm minimiert werden soll. Minimiert das Fenster, wodurch die Prozedur **`Form_Resize()`** aufgerufen wird, die die weitere Behandlung übernimmt.

4.1.3.26 Private Sub mn_Info_Click()

Parameter: Keine

Wird aus dem Kontextmenü des Traysymbols aufgerufen und zeigt den Info-Dialog **`dlgInfo`** an.

4.1.3.27 Private Sub SetColumnWidth()

Parameter: Keine

Diese Prozedur wird von mehreren Methoden aufgerufen. Sie stellt die Breite jeder einzelnen Spalte der Statusliste so ein, dass der komplette Text der Spalte angezeigt wird.

4.1.3.28 Private Sub TerminateProg()

Parameter: Keine

Diese Prozedur schreibt den Text „PMS_Transfer wird beendet“ ins Eventlog, entlädt das Fenster (nachdem **`m_bKilling`** auf **`True`** gestellt wurde, um unnötige Nachfragen zu vermeiden) und beendet das Programm.

4.1.3.29 Private Sub SetStatus()

Parameter: Optional ByVal s As String = ""

Diese Prozedur zeigt den angegebenen String in der Statusleiste an.

4.1.3.30 Private Sub LogStatus()

Parameter: *ByRef j As CJob*
ByVal sIcon As String

Übernimmt den aktuellen Status des Jobs und trägt ihn sowohl in der Statusliste, als auch in der Tabelle **[Srv_AutoTaskLog]** ein. Der Name des Icons wird dazu verwendet, das Symbol für die Statusliste anzugeben.

4.1.4 Öffentliche Funktionen

4.1.4.1 Public Function Form_Message() As Boolean

Parameter: *uMsg As Long*
wParam As Long
lParam As Long

Auch wenn diese Prozedur durch den Präfix **Form_** wie eine Systemprozedur aussieht, ist es doch eine Methode, die nur vom Modul **MultiHook** aufgerufen wird und zwar bei jeder Windows-Nachricht.

Hier wird nur eine Nachricht bearbeitet, und zwar **WM_GETMINMAXINFO**. Dies bedeutet, dass das System nach den minimalen und maximalen Abmessungen des Fensters fragt. In diesem Fall wird nur die Minimalgröße auf die Werte fixiert, die das Fenster beim ersten Öffnen hat.

Alle anderen Nachrichten werden ignoriert und mit **False** beantwortet. Dadurch wird dann im Modul **MultiHook** die Standardbehandlung aufgerufen.

4.1.5 Interne Funktionen

4.1.5.1 Private Function Status2Color() As String

Parameter: *ByVal nStat As atReturnConstants*

Diese Funktion übersetzt den Status eines Jobs in einen String, der wiederum ein Icon in der Imageliste bezeichnet. Das erleichtert das Füllen der Statusliste.

4.1.5.2 Private Function CheckJobs() As Boolean

Parameter: *Keine*

Diese Funktion wird von der Timerroutine aufgerufen, wenn die Wartezeit abgelaufen ist. Es werden die folgenden Aktionen durchgeführt:

Prüfen, ob **m_bKilling** gesetzt ist. Wenn ja, wird die Funktion sofort beendet, da das Programm die Aufforderung zum Beenden bekommen hat.

4.2 dlgLoginPMS

In diesem Dialog, der in identischer Form in den meisten PMS-Programmen existiert, können die Zugangsdaten zur PMS-Datenbank festgelegt werden.

ACHTUNG: Diese Einstellungen gelten für alle PMS-Programme, die auf diesem Rechner laufen! Eine Umstellung der Datenbank ist ein globaler Vorgang, der nicht einfach leichtfertig vorgenommen werden sollte!

Aus den hier festgelegten und in der Registrierung gespeicherten Angaben wird der Verbindungsstring erstellt, mit dem sich die Programme an der Datenbank anmelden.

Dazu gehören:

- Der Name oder die IP-Adresse des Rechners, auf dem der Datenbankserver läuft.
- Der Katalog (auch bekannt als Datenbank), der auf dem Datenbankserver angesprochen werden soll.
- Der Name des Benutzers, mit dem sich die PMS-Programme an der Datenbank anmelden sollen.
- Das Kennwort dieses Benutzers.

Beim Bestätigen des Dialogs prüft dieser, ob die Angaben gültig sind, indem er versucht, eine Datenbankverbindung mit diesen Angaben zu erstellen. Schlägt dies fehl, so bleibt der Dialog geöffnet und es wird eine entsprechende Warnung angezeigt.

Funktioniert die Verbindung, so werden die Angaben in der Registrierung gespeichert. Das bedeutet dann auch, dass jedes andere Programm, das seine Datenbankinformationen aus der Registrierung holt, nun auch die neuen Werte mitgeteilt bekommt.

4.2.1 Bildschirm-Elemente

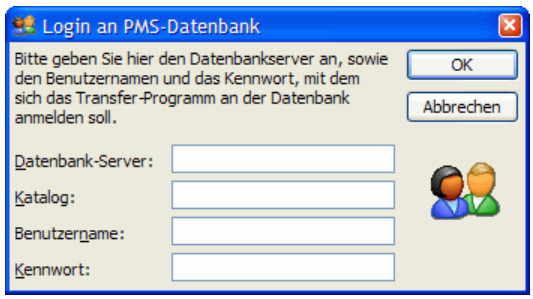

Abbildung 2: dlgLoginPMS

Elemente im Fenster:

- Eingabefeld **txtDbServer**. In diesem Feld wird der Name des Datenbankservers erwartet. Alle Buchstaben in diesem Feld werden in Großbuchstaben umgewandelt.
- Eingabefeld **txtCatalog**. In diesem Feld wird der Name der zu verwendenden Datenbank erwartet (z.B. **PMS**). Alle Buchstaben in diesem Feld werden in Großbuchstaben umgewandelt.
- Eingabefeld **txtUsername**. In diesem Feld wird der Anmeldename des Datenbankbenutzers erwartet.
- Eingabefeld **txtPwd**. In diesem Feld wird das zum Anmeldenamen passende Kennwort erwartet. Die hier eingegebenen Zeichen werden als **#** dargestellt.

- Button **OK**. Damit werden die Eingaben bestätigt.
- Button **Abbrechen**. Hiermit werden alle Änderungen verworfen und der Dialog geschlossen.

4.2.2 Variablen

4.2.2.1 Public bCommit As Boolean

Diese Variable wird beim Klicken von **Abbrechen** auf **False** und beim Klicken von **OK** auf **True** gestellt, sofern die eingegebenen Daten gültig sind und damit eine Verbindung zur Datenbank hergestellt werden konnte. Dadurch kann in der aufrufenden Routine der Status über diese Variable abgefragt werden.

4.2.3 Prozeduren

4.2.3.1 Private Sub Form_Load()

Parameter: Keine

Diese Prozedur wird beim Laden des Dialogs aufgerufen. Da der Dialog immer über **New** erzeugt wird, bedeutet das, dass die Methode bei der Initialisierung des Dialogs aufgerufen wird. Hier werden dann zuerst die Werte der verschiedenen Texte aus der Registrierung gelesen und in die Editierfelder eingetragen.

4.2.3.2 Private Sub ???_GotFocus()

Parameter: Keine

Für jedes der vier Eingabefelder **txtDbServer**, **txtCatalog**, **txtUsername** und **txtPwd** existiert hierbei eine eigene Prozedur. Diese wird jeweils dann aufgerufen, wenn das entsprechende Eingabefeld den Fokus erhält. Die Funktionalität der Prozeduren ist immer die gleiche: der komplette Text im Eingabefeld wird markiert.

4.2.3.3 Private Sub OKButton_Click()

Parameter: Keine

Diese Prozedur wird aufgerufen, wenn der Benutzer **OK** klickt oder die Eingabetaste drückt. Als erstes wird geprüft, ob die Eingabefelder **txtDbServer**, **txtUsername** oder **txtPwd** leer sind. Wenn ja, wird eine Fehlermeldung generiert, der Eingabefokus in das jeweilige Feld gesetzt und die Prozedur beendet. Dadurch bleibt der Dialog am Bildschirm und der Benutzer kann die benötigte Eingabe nachtragen.

Sind alle notwendigen Felder ausgefüllt, so wird nun versucht, mit diesen Angaben eine Verbindung zur Datenbank herzustellen. Schlägt dies fehl, springt die Prozedur in die Fehlerbehandlung (hinter dem Label **Hell**). Dort wird eine entsprechende Fehlermeldung generiert und die Prozedur beendet. Auch jetzt bleibt der Dialog am Bildschirm.

Funktioniert die (temporäre) Verbindung jedoch, so wird sie sofort wieder geschlossen und die eingegebenen Daten werden in der lokalen Registrierung gespeichert.

Danach wird **bCommit** auf True gestellt und der Dialog geschlossen.

4.2.3.4 Private Sub CancelButton_Click()

Parameter: Keine

Wird aufgerufen, wenn der Benutzer den Button **Abbrechen** klickt oder die ESC-Taste drückt.

Die Prozedur stellt **bCommit** auf False und schließt den Dialog einfach.

4.3 dlgLoginSAP

Mit diesem Dialog, der ähnlich wie **dlgLoginPMS** ausgelegt ist, können die Zugangsdaten zur SAP-Datenbank (eine Oracle-Datenbank) festgelegt werden.

Aus den hier festgelegten und in der Registrierung gespeicherten Angaben wird der Verbindungsstring erstellt, mit dem sich die Programme an der SAP-Datenbank anmelden.

Dazu gehören:

- Der TNS-Name zur Oracle-Datenbank, z.B. **„KOSA_P82“**.
- Der Name des Benutzers, mit dem sich das Programm an der Datenbank anmelden soll.
- Das Kennwort dieses Benutzers.

Beim Bestätigen des Dialogs prüft dieser, ob die Angaben gültig sind, indem er versucht, eine Datenbankverbindung mit diesen Angaben zu erstellen. Schlägt dies fehl, so bleibt der Dialog geöffnet und es wird eine entsprechende Warnung angezeigt.

Funktioniert die Verbindung, so werden die Angaben in der Registrierung gespeichert.

4.3.1 Bildschirm-Elemente

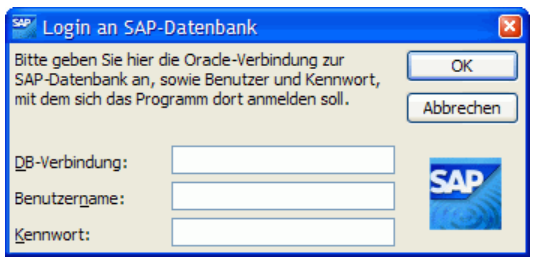

Abbildung 3: dlgLoginSAP

Elemente im Fenster:

- Eingabefeld **txtTnsName** (hinter **DB-Verbindung**). In diesem Feld wird der TNS-Name erwartet, über den sich das Oracle-Subsystem mit der Datenbank verbindet. Alle Buchstaben in diesem Feld werden automatisch in Großbuchstaben umgewandelt.
- Eingabefeld **txtUsername** (hinter **Benutzername**). In diesem Feld wird der Anmeldename des Datenbankbenutzers erwartet.
- Eingabefeld **txtPwd** (hinter **Kennwort**). In diesem Feld wird das zum Anmeldenamen passende Kennwort erwartet. Die hier eingegebenen Zeichen werden als **#** dargestellt.
- Button **OK**. Damit werden die Eingaben bestätigt und der Dialog geschlossen.
- Button **Abbrechen**. Hiermit werden alle Änderungen verworfen und der Dialog geschlossen.

4.3.2 Variablen

4.3.2.1 Public bCommit As Boolean

Diese Variable wird beim Klicken von **Abbrechen** auf **False** und beim Klicken von **OK** auf **True** gestellt, sofern die eingegebenen Daten gültig sind und damit eine Verbindung zur Datenbank hergestellt werden konnte. Dadurch kann in der aufrufenden Routine der Status über diese Variable abgefragt werden.

4.3.2.2 Private m_sTNSName As String

Interne Variable für den TNS-Namen.

4.3.2.3 Private m_sDbUser As String

Interne Variable für den Benutzernamen.

4.3.2.4 Private m_sDbPwd As String

Interne Variable für das Kennwort des Benutzers.

4.3.3 Interne Prozeduren

4.3.3.1 Private Sub Form_Initialize()

Parameter: Keine

Wird bei der Initialisierung des Fensters aufgerufen. Hier werden die internen Variablen aus der Registrierung gelesen (mit der Funktion **ConnectionString()**). Anschließend wird getestet, ob mit diesen Werten eine Verbindung zur Oracle-DB aufgebaut werden kann. Wenn ja, wird die Variable **bCommit** auf **True** gestellt.

Anm.: Die Funktionalität könnte auch problemlos nach **Form_Load()** übertragen werden.

4.3.3.2 Private Sub Form_Load()

Parameter: Keine

Diese Prozedur wird beim Laden des Dialogs aufgerufen. Da der Dialog immer über **New** erzeugt wird, bedeutet das, dass die Methode bei der Initialisierung des Dialogs aufgerufen wird. Hier werden dann die Werte der verschiedenen Texte aus der Registrierung gelesen und in die Editierfelder eingetragen und der Mauszeiger auf „Normal“ gestellt.

4.3.3.3 Private Sub OKButton_Click()

Parameter: Keine

Diese Prozedur wird aufgerufen, wenn der Benutzer **OK** klickt oder die Eingabetaste drückt. Als erstes wird geprüft, ob die Eingabefelder **txtTnsName**, **txtUsername** oder **txtPwd** leer sind. Wenn ja, wird eine Fehlermeldung generiert, der Eingabefokus in das jeweilige Feld gesetzt und die Prozedur beendet. Dadurch bleibt der Dialog am Bildschirm und der Benutzer kann die benötigte Eingabe nachtragen.

Sind alle notwendigen Felder ausgefüllt, so wird nun versucht, mit diesen Angaben eine Verbindung zur Oracle-Datenbank herzustellen. Schlägt dies fehl, springt die Prozedur in die Fehlerbehandlung (hinter dem Label **Hell**). Dort wird eine entsprechende Fehlermeldung generiert und die Prozedur beendet. Auch jetzt bleibt der Dialog am Bildschirm.

Funktioniert die (temporäre) Verbindung jedoch, so wird sie sofort wieder geschlossen und die eingegebenen Daten werden in der lokalen Registrierung gespeichert.

Danach wird **bCommit** auf True gestellt und der Dialog geschlossen.

4.3.3.4 Private Sub CancelButton_Click()

Parameter: Keine

Wird aufgerufen, wenn der Benutzer den Button **Abbrechen** klickt oder die ESC-Taste drückt.

Die Prozedur stellt **bCommit** auf False und schließt den Dialog einfach.

4.3.3.5 Private Sub CheckOK()

Parameter: Keine

Wird von den drei **Change()**-Methoden aufgerufen und prüft, ob alle Eingabefelder Text enthalten. Wenn nicht, wird der Button **OK** deaktiviert.

4.3.3.6 Private Sub ???_Change()

Parameter: Keine

Für jedes der drei Eingabefelder **txtTnsName**, **txtUsername** und **txtPwd** existiert hierbei eine eigene Prozedur, die jeweils die Methode **CheckOK()** aufruft.

4.3.3.7 Private Sub ???_GotFocus()

Parameter: Keine

Für jedes der drei Eingabefelder **txtTnsName**, **txtUsername** und **txtPwd** existiert hierbei eine eigene Prozedur. Diese wird jeweils dann aufgerufen, wenn das entsprechende Eingabefeld den Fokus erhält. Die Funktionalität der Prozeduren ist immer die gleiche: der komplette Text im Eingabefeld wird markiert (um das normale Verhalten von Windows-Programmen zu erzeugen).

4.3.3.8 Private Sub ???_KeyPress()

Parameter: KeyAscii As Integer

Existiert für die Eingabefelder **txtTnsName** und **txtUsername**. Mit diesen Methoden werden die Eingaben in Gr0ßbuchstaben umgewandelt. Beim Kennwort darf das natürlich nicht sein!

4.3.4 Interne Funktionen

4.3.4.1 Private Function CheckConnection() As Boolean

Parameter: Keine

Diese Funktion versucht, mit den gegebenen Daten eine Verbindung zur SAP-Datenbank aufzubauen. Dazu wird zuerst der Verbindungs-String mit der Funktion **ConnectionString()** geholt. Ist dieser String leer, bricht die Funktion mit dem Ergebnis **False** ab. Sonst wird eine neue **ADODB-Connection** erzeugt, diese initialisiert und geöffnet. Funktioniert die Verbindung, so liefert die Funktion **True** zurück, ansonsten **False**.

4.3.5 Öffentliche Funktionen

4.3.5.1 Public Function ConnectionString() As String

Parameter: Optional ByVal bReload As Boolean = False

Diese Funktion verwendet die internen Variablen **m_sTNSName**, **m_sDbUser** und **m_sDbPwd**, um einen gültigen Verbindungsstring zu erstellen, mit dem eine Verbindung zur SAP-Datenbank hergestellt werden kann. Der String sieht folgendermaßen aus:

```
„Provider=MSDAORA.1;Data Source=TNS-Name;User ID=Benutzer;Password=Kennwort;“
```

Code 5: Oracle-Verbindungsstring

Fehlt einer der drei notwendigen Teile, so wird ein Leerstring geliefert. Es wird hier nicht geprüft, ob mit den Angaben eine Verbindung hergestellt werden kann!

4.4 dlgMsg

Dieser Dialog wird verwendet, um Nachrichten (hauptsächlich Fehlermeldungen) anzuzeigen. Das Besondere daran ist, dass der Dialog sich nach 5 Sekunden automatisch schließt. Das ist notwendig, da das Programm normalerweise dafür gedacht ist, auf einem Rechner zu laufen, der nicht ständig von Personal überwacht wird.

Die Fehlermeldungen gehen dabei nicht verloren, da sie außerdem noch in der Datenbank und / oder im lokalen Eventlog abgelegt werden.

Ein Beispiel, wie dieser Dialog im Programm (Prozedur **ErrorHandler**) verwendet wird:

```
dlgMsg.Msg = sMsg       ' Fehlertext an Dialog zuweisen
dlgMsg.Show vbModal     ' Dialog anzeigen
Unload dlgMsg           ' Dialog entladen
```

Tabelle 14: dlgMsg aufrufen

Durch die Zuweisung des Textes an die Variable **Msg** wird der Dialog implizit geladen. Anschließend wird er angezeigt und nach dem Schließen (entweder durch **OK** oder durch die Automatik) wieder entladen.

4.4.1 Bildschirm-Elemente

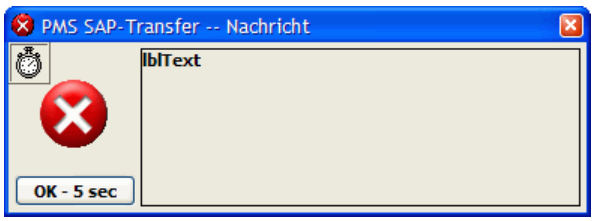

Abbildung 4: dlgMsg

Elemente im Fenster:

- Textfeld **lblText**. In diesem Feld wird die anzuzeigende Nachricht dargestellt.
- Timer **tim**. Ein Timer, der jede Sekunde ein Ereignis auslöst.
- Schaltfläche **OK-? sec**. Wird sie gedrückt, schließt sich der Dialog auch schon vor dem Ablauf der Zeit. Außerdem wird hier die noch verbleibende Zeit bis zum automatischen Schließen angezeigt.
- Symbol **imgStop**. Eine einfache Grafik, die ein übergroßes Stoppsignal darstellt.

4.4.2 externe API-Deklarationen

4.4.2.1 Private Declare Function MessageBeep Lib "user32" () As Long

Parameter: *ByVal wType As Long*

Mit dieser Methode wird einfach ein Warnsignal erzeugt, wenn der Dialog angezeigt wird. Der Parameter **wType** ist einfach der gewünschte Ton (hier wird immer **MB_ICONHAND** (**&H10**) verwendet).

4.4.3 Variablen

4.4.3.1 Private m_Time As Long

Diese Variable enthält die Anzahl der Sekunden, die der Dialog angezeigt werden soll. Sie steht per Default auf 5, kann jedoch nach Geschmack im Code verändert werden. Sie wird im Timer-Event jeweils um eins heruntergezählt.

4.4.3.2 Public Msg As String

Der Text, der in diesem String steht, wird — als Alarm oder Hinweis, etc. — angezeigt. Die Zuweisung des Textes muss vor der Anzeige des Dialogs geschehen, eine spätere Zuweisung ist sinnlos[17].

4.4.4 Interne Prozeduren

4.4.4.1 Private Sub Form_Load()

Beim Laden des Dialogs (entweder explizit oder implizit) wird der interne Timer auf 1000ms gestellt und der Sekundenzähler auf 5. Danach wird der Timer aktiviert und der Text im Feld **`lblText`** eingetragen.

Zum Schluss wird noch ein Alarmsignal erzeugt, dessen Ton von der allgemeinen Systemeinstellung abhängig ist.

4.4.4.2 Private Sub tim_Timer()

Diese Prozedur wird nach jedem Ablauf des internen Timers aufgerufen. Hier wird der Sekundenzähler um 1 verringert und die Restzeit als Text auf der Schaltfläche angezeigt.

Ist die Zeit (nach 5 Sekunden) abgelaufen, so wird die Prozedur **`OKButton_Click()`** aufgerufen.

4.4.4.3 Private Sub OKButton_Click()

Diese Prozedur wird entweder durch das Klicken der **OK**-Schaltfläche oder durch den abgelaufenen Timer aufgerufen.

Hier wird der Timer deaktiviert und das Fenster versteckt, wodurch der Dialog[18] beendet wird.

[17] ...und im Normalfall auch gar nicht möglich, da der Dialog modal angezeigt wird und daher die Kontrolle erst dann wieder abgibt, wenn er geschlossen wird.

[18] das funktioniert auch nur unter VisualBasic so!

4.5 dlgInfo

Dieser Dialog dient nur zur Information. Er zeigt an, wie die verschiedenen Wartezeiten der einzelnen Jobs eingestellt sind. Es gibt hier keine Möglichkeit, etwas zu ändern — dazu sind die verschiedenen Möglichkeiten in **`frmMain`** vorgesehen.

4.5.1 Bildschirm-Elemente

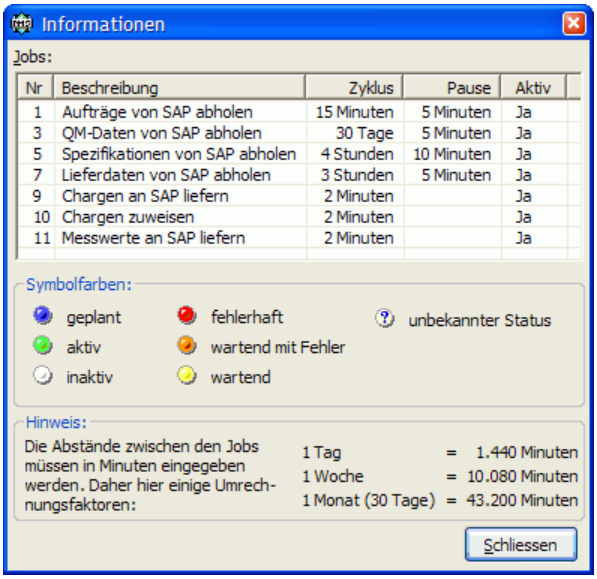

Abbildung 5: dlgInfo

Elemente im Fenster:

- Liste der Jobs. In dieser Liste werden alle bekannten Jobs, sowie die zugehörigen Wartezeiten aufgelistet.
- Symbolfarben. Im Hauptfenster wird der Status der Jobs durch kleine farbige Symbole dargestellt. Deren Bedeutung wird hier erklärt.
- Hinweis. Da die Wartezeiten bei der Eingabe in Minuten anzugeben sind, folgt hier noch ein kurzes Info-Feld mit einigen Umrechnungsfaktoren.
- Button **Schliessen**. Er schließt das Fenster.

4.5.2 Interne Prozeduren

4.5.2.1 Private Sub Form_Load()

Parameter: Keine

Beim Laden des Dialogs werden aus der Datenbank die definierten Jobs geladen. Aus diesen Daten wird dann die Liste aufgebaut. Dabei werden die Zeitspannen für den Zyklus und die Pause mit der Funktion **`Sec2Str()`** in lesbaren Text umgewandelt.

Zuletzt werden noch die Spaltenbreiten der Liste so angepasst, dass alle Texte komplett dargestellt werden.

4.5.2.2 Private Sub m_OK_Click()

Parameter: Keine

Wird dann aufgerufen, wenn der Button **Schliessen** gedrückt oder **Esc** gedrückt wird. Hier wird einfach der Dialog entladen, wodurch er natürlich auch vom Bildschirm verschwindet.

4.5.3 Interne Funktionen

4.5.3.1 Private Function Sec2Str() As String

Parameter: ByVal n As Long

Zur besseren Anzeige der Zeitspannen werden die Sekundenwerte hier in lesbaren Text umgewandelt. Dabei wird die Lesbarkeit höher bewertet als die Präzision; d.h. es werden nur ganze Werte angezeigt!

Beispiele:

`25`: "25 Sekunden", **`125`**: "2 Minuten", **`3600`**: "1 Stunden", **`4000`**: "1 Stunden", **`7500`**: "2 Stunden", **`86400`**: "1Tage"

5 Klassen

PMS_Party arbeitet mit einigen Klassen, in denen Daten (teilweise) sinnvoll voneinander getrennt werden.

5.1 CJob

Diese Klasse enthält die eigentlichen Arbeitsroutinen des Programms. Je nach Einstellung verarbeitet sie die entsprechenden Jobs in der Routine **`Execute()`**. Diese wird vom Hauptfenster über die Timerroutine aufgerufen.

Damit der entsprechende Job richtig funktioniert, muss eine gewisse Reihenfolge beim Erstellen eingehalten werden:

1. Job anlegen (**`Set Job = New CJob`**)
2. Typ definieren (**`Job.Typ = ...`**)
3. Zeit festlegen (**`Job.RunTime = ...`**)
4. Minimale Zeit zwischen den Einzelaufrufen festlegen (**`Job.Delay = ...`**)
5. bei 2-stufigen Jobs die Zeit zwischen den Schritten festlegen (**`Job.StepTwo = ...`**)
6. Aktivieren (**`Job.Active = True`**)
7. Job ausführen lassen, z.B. in Timer-Event (**`Job.Execute sMsg`**)

5.1.1 Datentypen und Enums

```
Public Enum atTypes
   atTypeUnknown              = 0
   atTypeOrderGet             = 1
   atTypeOrderRequest         = 2
   atTypeQmBatchGet           = 3
   atTypeQmBatchRequest       = 4
   atTypeSpecGet              = 5
   atTypeSpecRequest          = 6
   atTypeDeliverDataGet       = 7
   atTypeDeliverDataRequest   = 8
   atTypeChargesPut           = 9
   atTypeChargesAssign        = 10
   atTypeValuesPut            = 11
   atTypeInvalid              = 12
End Enum
```

Tabelle 15: Enum atTypes

Dieser Datentyp dient zur Unterscheidung und Festlegung der einzelnen Jobs. Die Typen, die mit „**`Get`**“ und „**`Request`**“ enden, stellen die zwei Stufen bei den SAP-Anforderungen dar.

```
Public Enum atReturnConstants
   atrSuccess         = 0
   atrWait            = -1
   atrFailed          = -2
   atrFailedWaiting   = -3
   atrPlanned         = -4
   atrWorking         = -5
   atrInactive        = -6
End Enum
```

Tabelle 16: Enum atReturnConstants

Dieser Datentyp stellt die möglichen Ergebnisse dar, die beim Ausführen der einzelnen Methoden zurückgeliefert werden.

5.1.2 Interne Variablen / Eigenschaften

Alle Zustände und Daten der Klasse werden in privaten Variablen gehalten. Einige davon werden nur intern bearbeitet und sind daher von außen gar nicht sichtbar, andere werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_Active	**Boolean**	**Active**	R/W	Enthält **`True`**, wenn der Job aktiviert ist, sonst **`False`**.
m_Delay	**Long**	**Delay**	R/W	Die Zeit in Minuten, die mindestens zwischen zwei Ereignissen liegen muss.
m_Desc	**String**	**Description**	R	Die Bezeichnung des Jobs als Klartext.
m_RunTime	**Date**	**RunTime**	R/W	Der Zeitpunkt der nächsten Ausführung. Nachdem ein Job ausgeführt wurde, wird der Wert von **`m_Delay`** zur aktuellen Zeit addiert, um den nächsten Ausführungszeitpunkt festzulegen.
m_StateText	**String**	**StatusText**	R	Der aktuelle Status des Jobs in Textform.
m_StateVal	**atReturnConstants**	**Status**	R	Der aktuelle Status des Jobs in numerischer Form.
m_Step	**Integer**	**---**		Bei mehrstufigen Jobs (Auftrag, QM-Daten, Spezifikationen und Lieferdaten) enthält diese Variable **1** für die erste Stufe (Anfordern) oder **2** für die zweite Stufe (Abholen).
m_StepTwo	**Long**	**WaitTime**	R/W	Die Zeit in Sekunden, die bei mehrstufigen Jobs (Auftrag, QM-Daten, Spezifikationen und Lieferdaten) zwischen der ersten und der zweiten Stufe mindestens gewartet wird[19].
m_Type	**atTypes**	**Typ**	R/W	Der Typ des Jobs. Möglich sind hierbei alle Werte, die im Enum **`atTypes`** definiert sind.

Tabelle 17: CJob, interne Variablen

[19] Die tatsächliche Wartezeit ergibt sich auch aus der Aufruffrequenz: Je häufiger die Methode **`Execute()`** aufgerufen wird, desto präziser stimmen die hier definierten Zeitspannen. Nur durch diese Methode werden die einzelnen Jobs tatsächlich ausgeführt.

5.1.3 Interne Prozeduren

5.1.3.1 Private Sub Class_Initialize()

Parameter: Keine

Wird bei der Erstellung der Klasse (z.B. durch **New**) aufgerufen. Ruft **Clear()** auf und stellt **m_RunTime** auf 5 Sekunden nach der aktuellen Zeit.

5.1.3.2 Private Sub IncRuntime()

Parameter: Keine

Addiert die Minuten in **m_Delay** zum Ausführungszeitpunkt. Dabei wird sichergestellt, dass der neue Zeitpunkt nicht in der Vergangenheit liegt.

5.1.4 Interne Funktionen

5.1.4.1 Private Function SAP_ChargenZuordnen() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeChargesAssign** ist. Diese Methode hat eigentlich noch nichts mit SAP zu tun, vielmehr werden die Chargen hiermit den entsprechenden Aufträgen zugeordnet.

Dazu werden in der PMS-Datenbank, Tabelle **[Chargen]** alle Chargen gesucht, die noch nicht versendet wurden (Feld **[Status_Work]** enthält "**0**" oder "**A**"). Existieren keine neuen Chargen, so wird die Funktion mit dem Ergebnis „Warten“ (**atrWait**) beendet.

Existieren dagegen solche Chargen, so wird versucht, über System und Materialnummer passende, aktive[20] Aufträge in der Tabelle **[Auftrag]** zu finden. Existiert ein solcher Auftrag, wird seine Auftragsnummer im Datensatz der Chargentabelle eingetragen.

Zum Abschluss wird der String in **sMessage** noch mit einem erläuternden Text gefüllt und die Funktion mit dem Ergebnis „Erfolg“ (**atrSuccess**) beendet.

5.1.4.2 Private Function SAP_ChargenSenden() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeChargesPut** ist. Hier werden alle abgeschlossenen, aber noch nicht übertragenen Chargen (Feld **[Status_Work]** enthält "**0**" oder "**A**") gesucht und mit den Methoden der Klasse **CSapCharge** an SAP übertragen.

Abschließend wird SAP mit der Methode **SAPAktionSenden(sapU_CHARGE_VP)** der Klasse **CSapGlobals** noch mitgeteilt, dass neue Chargen abzuholen sind.

Danach wird das Feld **[Status_Work]** aller übertragenen Chargen in der PMS-Tabelle **[Chargen]** auf „gesendet“ gestellt. Dazu werden Datensätze, deren Status auf "**0**" steht, auf "**2**" gestellt und solche, deren Status auf "**A**" steht, auf "**B**" gestellt.

[20] Ein aktiver Auftrag ist dadurch gekennzeichnet, dass sein Feld **[Begin]** einen Zeitpunkt vor „Jetzt“ und sein Feld **[Ende] NULL** enthält.

Zum Abschluss wird der String in **sMessage** noch mit der Anzahl der versendeten Chargen gefüllt und die Funktion mit dem Ergebnis „Erfolg“ (**atrSuccess**) beendet.

5.1.4.3 Private Function SAP_MesswerteSenden() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeValuesPut** ist. Zuerst werden aus der PMS-Tabelle **[QmDaten]** die Messwerte gesucht, die noch nicht an SAP übertragen wurden. Sind keine vorhanden, so wird die Funktion mit dem Ergebnis „Wartend“ (**atrWait**) beendet.

Ansonsten werden alle Datensätze, die ein gültiges Merkmal enthalten, mit den Methoden der Klasse **CSapMesswert** an SAP übertragen.

Abschließend wird SAP mit der Methode **SAPAktionSenden(sapU_MESSWERT_VP)** der Klasse **CSapGlobals** noch mitgeteilt, dass neue Messwerte abzuholen sind.

Danach wird das Feld **[Status_Work]** aller übertragenen Messwerte in der PMS-Tabelle **[QmDaten]** auf „gesendet“ gestellt. Dazu werden Datensätze, deren Status auf "**0**" steht, auf "**2**" gestellt und solche, deren Status auf "**A**" steht, auf "**B**" gestellt.

5.1.4.4 Private Function SAP_AuftragAnfordern() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeOrder...**[21] und **m_Step 1** ist. Hier wird nur die Funktion **SAPAktionSenden(sapD_AUF_VP)** des Objekts **g_SAP** aufgerufen. Dadurch wird in der SAP-Tabelle **ZAKTION_VP_4B** der Wert "**D_AUF_VP_4B**" eingetragen. Dieser Eintrag fordert SAP dazu auf, neue Aufträge zu ermitteln und diese dann in der SAP-Tabelle **ZAUF_VP_4B** bereitzustellen.

Hat das Absenden des Kommandos funktioniert, liefert die Funktion „Warten“ (**atrWait**) als Ergebnis, ansonsten „Fehlgeschlagen“ (**atrFailed**). Außerdem wird der Statustext im Parameter **sMessage** noch entsprechend eingestellt.

Nach dem erfolgreichen Ausführen dieser Funktion wechselt die Stufe (in der Funktion **Execute()**) von 1 auf 2. Dadurch wird als nächster Job die folgende **SAP_AuftragHolen()** ausgeführt.

[21] Dies kann **atTypeOrderGet** oder **atTypeOrderRequest** sein. Der Unterschied liegt hierbei nur im Wert von **m_Step**.

5.1.4.5 Private Function SAP_AuftragHolen() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeOrder...** und **m_Step 2** ist. Dies ist dann der Fall, wenn das Anfordern von Aufträgen in **SAP_AuftragAnfordern()** erfolgreich war. Nachdem die Wartezeit zwischen Stufe 1 und 2 abgelaufen ist, wird mit dieser Methode nachgesehen, ob SAP in der Tabelle **ZAUF_VP_4B** neue Aufträge (mit dem Übertragungsstatus 4) zur Verfügung gestellt hat. Wenn das nicht der Fall ist, wird die Funktion mit dem Status „Warten" (**atrWait**) beendet[22].

Sind neue Aufträge vorhanden, so werden sie einzeln in der PMS-Tabelle **[Auftrag]** gesucht. Sind sie dort bereits vorhanden, wird nichts verändert. Im anderen Fall werden sie neu angelegt.

Nachdem alle gelieferten Aufträge abgearbeitet sind, werden in der SAP-Tabelle **ZAUF_VP_4B** alle Datensätze mit Status 4 (= zu verarbeiten) auf Status 6 (= verarbeitet) gesetzt.

Zuletzt wird in **sMessage** noch ein Text zusammengestellt, der die Anzahl der übertragenen sowie die Anzahl der importierten Aufträge enthält und die Funktion mit dem Status „Erfolg" (**atrSuccess**) beendet.

5.1.4.6 Private Function SAP_LieferdatenAnfordern() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeDeliverData...**[23] und **m_Step 1** ist. Hier wird nur die Funktion **SAPAktionSenden(sapD_LIEFER_VP)** des Objekts **g_SAP** aufgerufen. Dadurch wird in der Tabelle **ZAKTION_VP_4B** der Wert "**D_LIEFER_VP_4B**" eingetragen. Dieser Eintrag fordert SAP dazu auf, neue Lieferdaten zu ermitteln und diese dann in der SAP-Tabelle **ZQM_LIEFER_VP_4B** bereitzustellen.

Hat das Absenden des Kommandos funktioniert, liefert die Funktion „Warten" (**atrWait**) als Ergebnis, ansonsten „Fehlgeschlagen" (**atrFailed**). Außerdem wird der Statustext im Parameter **sMessage** noch entsprechend eingestellt.

Nach dem erfolgreichen Ausführen dieser Funktion wechselt die Stufe (in der Funktion **Execute()**) von 1 auf 2. Dadurch wird als nächstes das folgende **SAP_LieferdatenHolen()** ausgeführt.

5.1.4.7 Private Function SAP_LieferdatenHolen() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeDeliverData...** und **m_Step 2** ist und das Anfordern von Lieferdaten in **SAP_LieferdatenAnfordern()** erfolgreich war. Nachdem die Wartezeit zwischen Stufe 1 und 2 abgelaufen ist, wird mit dieser Methode nachgesehen, ob SAP in der Tabelle **ZQM_LIEFER_VP_4B** neue Lieferdaten (mit dem Übertragungsstatus 4) zur Verfügung gestellt hat. Wenn das nicht der Fall ist, wird die Funktion mit dem Status „Warten" (**atrWait**) beendet.

---

[22] Sollte SAP zum Ausführen mehr Zeit benötigen, als in der Verzögerung festgelegt ist, können die Daten nicht sofort übertragen werden. Dies ist aber unkritisch, da die in der Transfertabelle abgelegten Daten nicht verloren gehen, sondern im nächsten Durchgang gelesen. Bei der üblichen Einstellung des Systems ergibt sich daraus eine zusätzliche Verzögerung von ca. 5 Minuten, die aber völlig unkritisch ist.

[23] Dies kann **atTypeDeliverDataGet** oder **atTypeDeliverDataRequest** sein. Der Unterschied liegt hierbei nur im Wert von **m_Step**.

Sind neue Lieferdaten vorhanden, so werden sie einzeln in der PMS-Tabelle **[Lieferdaten]** gesucht. Sind sie dort bereits vorhanden, wird nichts verändert. Im anderen Fall werden sie neu angelegt.

Nachdem alle gefundenen Lieferdaten abgearbeitet sind, werden in der SAP-Tabelle **ZQM_LIEFER_VP_4B** alle Datensätze mit Status 4 (= zu verarbeiten) auf Status 6 (= verarbeitet) gesetzt.

Zuletzt wird in **sMessage** noch ein Text zusammengestellt, der die Anzahl der übertragenen sowie die Anzahl der importierten Lieferdaten enthält und die Funktion mit dem Status „Erfolg“ (**atrSuccess**) beendet.

5.1.4.8 Private Function SAP_QMDatenAnfordern() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeQmBatch** und **m_Step 1** ist. Hier wird nur die Funktion **SAPAktionSenden(sapD_BATCH_VP)** des Objekts **g_SAP** aufgerufen. Dadurch wird in der Tabelle **ZAKTION_VP_4B** der Wert "**D_BATCH_VP_4B**" eingetragen. Dieser Eintrag fordert SAP dazu auf, neue Qualitätsmerkmale zu ermitteln und diese dann in der SAP-Tabelle **ZQM_BATCH_VP_4B** bereitzustellen.

Hat das Absenden des Kommandos funktioniert, liefert die Funktion „Warten“ (**atrWait**) als Ergebnis, ansonsten „Fehlgeschlagen“ (**atrFailed**). Außerdem wird der Statustext im Parameter **sMessage** noch entsprechend eingestellt.

Nach dem erfolgreichen Ausführen dieser Funktion wechselt die Stufe (in der Funktion **Execute()**) von 1 auf 2. Dadurch wird als nächstes das folgende **SAP_QMDatenHolen()** ausgeführt.

5.1.4.9 Private Function SAP_QMDatenHolen() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeQmBatch** und **m_Step 2** ist und das Anfordern von Qualitätsmerkmalen in **SAP_QMDatenAnfordern()** erfolgreich war. Nachdem die Wartezeit zwischen Stufe 1 und 2 abgelaufen ist, wird mit dieser Methode nachgesehen, ob SAP in der Tabelle **ZQM_BATCH_VP_4B** neue Qualitätsmerkmale (mit dem Übertragungsstatus 4) zur Verfügung gestellt hat. Wenn das nicht der Fall ist, wird die Funktion mit dem Status „Warten“ (**atrWait**) beendet.

Sind neue Qualitätsmerkmale vorhanden, so werden sie einzeln in der PMS-Tabelle **[QmDaten]** gesucht. Sind sie dort bereits vorhanden, werden nur die Felder **[Wert]**, **[Num_wert]** und **[Num_Einh]** angepasst. Im anderen Fall wird ein kompletter Datensatz neu angelegt.

Nachdem alle gefundenen Qualitätsmerkmale abgearbeitet sind, werden in der SAP-Tabelle **ZQM_BATCH_VP_4B** alle Datensätze mit Status 4 (= zu verarbeiten) auf Status 6 (= verarbeitet) gesetzt.

Zuletzt wird in **sMessage** noch ein Text zusammengestellt, der die Anzahl der übertragenen sowie die Anzahl der importierten Qualitätsmerkmale enthält und die Funktion mit dem Status „Erfolg“ (**atrSuccess**) beendet.

5.1.4.10 Private Function SAP_SpezifikationAnfordern() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeSpec** und **m_Step 1** ist. Hier wird nur die Funktion **SAPAktionSenden(sapD_SPEZ_VP)** des Objekts **g_SAP** aufgerufen. Dadurch wird in der Tabelle **ZAKTION_VP_4B** der Wert "**D_SPEZ_VP_4B**" eingetragen. Dieser Eintrag fordert SAP dazu auf, neue Spezifikationen zu ermitteln und diese dann in der SAP-Tabelle **ZSPEZ_VP_4B** bereitzustellen.

Hat das Absenden des Kommandos funktioniert, liefert die Funktion „Warten“ (**atrWait**) als Ergebnis, ansonsten „Fehlgeschlagen“ (**atrFailed**). Außerdem wird der Statustext im Parameter **sMessage** noch entsprechend eingestellt.

Nach dem erfolgreichen Ausführen dieser Funktion wechselt die Stufe (in der Funktion **Execute()**) von 1 auf 2. Dadurch wird als nächstes **SAP_SpezifikationHolen()** ausgeführt.

5.1.4.11 Private Function SAP_SpezifikationHolen() As atReturnConstants

Parameter: ByRef sMessage As String

Wird von **Execute()** aufgerufen, wenn der Typ des Jobs **atTypeSpec** und **m_Step 2** ist und das Anfordern von Spezifikationen in **SAP_SpezifikationAnfordern()** erfolgreich war. Nachdem die Wartezeit zwischen Stufe 1 und 2 abgelaufen ist, wird mit dieser Methode nachgesehen, ob SAP in der Tabelle **ZSPEZ_VP_4B** neue Spezifikationen (mit dem Übertragungsstatus 4) zur Verfügung gestellt hat. Wenn das nicht der Fall ist, wird die Funktion mit dem Status „Warten“ (**atrWait**) beendet.

Sind neue Spezifikationen vorhanden, so werden sie einzeln in der PMS-Tabelle **[QmDaten]** gesucht. Sind sie dort bereits vorhanden, wird nichts verändert. Im anderen Fall wird ein entsprechender Datensatz neu angelegt.

Nachdem alle gefundenen Datensätze abgearbeitet sind, werden in der Tabelle **ZSPEZ_VP_4B** alle Datensätze mit Status 4 (= zu verarbeiten) auf Status 6 (= verarbeitet) gesetzt.

Zuletzt wird in **sMessage** noch ein Text zusammengestellt, der die Anzahl der übertragenen sowie die Anzahl der importierten Spezifikationen enthält und die Funktion mit dem Status „Erfolg“ (**atrSuccess**) beendet.

5.1.4.12 Private Function UpdateField() As Boolean

Parameter: ByRef fld As ADODB.Field
ByVal sValue As String

Schreibt geänderte Werte in das Feld **fld** im Recordset zurück. Das Feld wird nur bei unterschiedlichen Daten überschrieben, leere Strings werden zuvor in **NULL** konvertiert. Die Funktion gibt **True** zurück, wenn der Wert sich gegenüber dem vorigen Zustand geändert hat, sonst **False**.

5.1.4.13 Private Function ConvSAPDate() As Date

Parameter: *ByVal sDate As String*
Optional ByVal sTime As String = ""

Konvertiert einen Datumsstring aus dem in SAP verwendeten Format "**YYYYMMDD**" in eine normale Datumsvariable. Sofern angegeben, wird auch noch die Uhrzeit aus dem Format "**HHMMSS**" konvertiert und dazu addiert. Enthält der Datumsstring nicht mindestens 8 Zeichen, so wird als Ergebnis **0** geliefert. Bei der Uhrzeit wird ein String von mindestens 6 Zeichen erwartet. Bei kürzeren Strings wird die Uhrzeit ignoriert.

5.1.5 Öffentliche Prozeduren

5.1.5.1 Public Sub Clear()

Parameter: *Keine*

Initialisiert die Klasse auf den Ausgangszustand. Alle Variablen werden dabei auf Standardwerte gesetzt, das Objekt hat keinen Typ und ist inaktiv.

5.1.6 Öffentliche Funktionen

5.1.6.1 Public Function Load() As Boolean

Parameter: *ByVal t As atTypes*

Lädt die Einstellungen für den angegebenen Typ aus der Datenbank. Danach wird der Status auf „geplant“ (**atrPlanned**) und die aktuelle Stufe auf 1 gesetzt.

5.1.6.2 Public Function Save() As Boolean

Parameter: *Keine*

Speichert die internen Daten in der Datenbank. Dabei werden der Zeitpunkt der nächsten Ausführung, die Zeitspanne zwischen zwei Ausführungen, die aktuelle Schrittnummer (1 oder 2), sowie der Aktivierungsstatus gespeichert.

5.1.6.3 Public Function Execute() As Boolean

Parameter: *Keine*

Dies ist die wichtigste Funktion der Klasse. Mit ihr wird der eigentliche Auftrag des Objekts, abhängig vom eingestellten Typ, ausgeführt.

Zuerst wird kontrolliert, ob der Status „aktiv“ ist. Wenn nicht, werden Statuswert und Statustext eingestellt und die Funktion mit dem Ergebnis **False** verlassen.

Als nächstes wird getestet, ob der Zeitpunkt zur Ausführung bereits erreicht ist. Wenn nicht, wird der Status auf „geplant“ (**atrPlanned**) gesetzt und die Funktion ebenfalls mit **False** verlassen.

Danach wird — abhängig von Typ und (bei mehrstufigen Jobs) der Stufe — eine der internen Arbeitsfunktionen, die mit **SAP**_... beginnen, aufgerufen. Diese liefern ihre Ergebnisse in den Statusvariablen zurück. Sollte der Typ unbekannt oder ungültig sein, wird der Status auf „fehlgeschlagen“ (**atrFailed**) gesetzt und als Statustext eine entsprechende Fehlermeldung generiert.

Zum Abschluss wird noch der Zeitpunkt der nächsten Ausführung ermittelt, bzw. wenn der Job in der Stufe ist, der Status auf „in Arbeit“ (**atrWorking**) gesetzt. Bei erfolgreich ausgeführten Jobs wird der Ergebniswert auf **True** gesetzt.

Danach wird die Funktion beendet.

5.1.6.4 Public Function IsTime() As Boolean

Parameter: Keine

Diese Funktion ermittelt, ob der Job jetzt ausgeführt werden muss. Dazu wird die geplante Laufzeit mit der aktuellen Uhrzeit verglichen. Ist der geplante Zeitpunkt noch nicht erreicht, liefert die Funktion **False** zurück.

Ansonsten wird bei zweistufigen Jobs kontrolliert, ob Stufe 2 aktuell ist und die Pause zwischen den Stufen bereits abgelaufen ist. Ist Stufe 1 aktuell, wird auf jeden Fall **True** geliefert. Ansonsten wird solange **False** geliefert, wie die Pausenzeit noch nicht abgelaufen ist.

5.2 CSapCharge

Diese Klasse verwaltet eine Charge in dem Format, wie sie zum SAP transportiert wird. Sie bietet Methoden, um die Chargendaten in die SAP-Transfertabellen zu übertragen.

Verwendet wird die Klasse von **CJob** und zwar ausschließlich beim Senden von Chargen (Funktion **SAP_ChargenSenden()**). Dort werden jeweils lokale Objekte der Klasse angelegt.

5.2.1 Interne Variablen / Eigenschaften

Alle Zustände und Daten der Klasse werden in privaten Variablen gehalten. Einige davon werden nur intern bearbeitet und sind daher von außen gar nicht sichtbar, andere werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_MANDT	**Long**	**---**	---	Die Mandantennummer zur Anmeldung im SAP. Diese Nummer wird aus dem globalen Objekt **g_SAP** übernommen[24].
m_WERK	**String**	**---**	---	Das Werkskürzel im SAP[25]. Diese Angabe wird aus dem globalen Objekt **g_SAP** übernommen.
m_TID	**String**	**---**	---	Die Transaktions-ID für SAP. Diese Angabe wird aus dem globalen Objekt **g_SAP** übernommen.
m_DATUM	**Date**	**---**	---	Datum und Uhrzeit der Erstellung des Objekts. Wird beim Senden an SAP übertragen.
m_ZEIT	**Date**			
m_ISDD	**Date**	**ISDD**	R/W	Das Startdatum der Charge.
m_ISDZ	**Date**	**ISDZ**	R/W	Die Startzeit der Charge.
m_IEDD	**Date**	**IEDD**	R/W	Das Enddatum der Charge.
m_IEDZ	**Date**	**IEDZ**	R/W	Die Endzeit der Charge.
m_AUFNR	**String**	**AUFNR**	R/W	Die Nummer des Auftrags, zu dem die Charge gehört.
m_MATNR	**String**	**MATNR**	R/W	Die Materialnummer der Charge.
m_CHARG	**String**	**CHARG**	R/W	Die Bezeichnung der Charge.
m_ERFMG	**String**	**ERFMG**	R/W	Das Gewicht der Charge.
m_LGPLA	**String**	**LGPLA**	R/W	Der Lagerplatz[26] der Charge.
m_Status	**String**	**Status**	R/W	Der Status der Charge, wie er an SAP übertragen wird.

Tabelle 18: CSapCharge, interne Variablen

[24] Hierbei besteht eine Diskrepanz der Datentypen! In **CSapCharge** wird der Mandant als Long gehalten, in **g_SAP** als String. Da der Inhalt jedoch numerisch ist, stellt das zurzeit kein Problem dar, sollte jedoch im Auge behalten werden.

[25] Dieses lautete für das Werk Offenbach **„4B“**

[26] Tatsächlich wird damit eher das System bezeichnet, in dem die Charge hergestellt wurde.

5.2.2 Interne Prozeduren

5.2.2.1 Private Sub Class_Initialize()

Parameter: Keine

Wird bei der Erstellung der Klasse (z.B. durch **New**) aufgerufen. Ruft die interne Prozedur **Clear()** auf.

5.2.3 Interne Funktionen

5.2.3.1 Private Function Update() As Boolean

Parameter: ByRef con As ADODB.Connection

Wird von der Funktion **Send()** aufgerufen, wenn das Einfügen einer neuen Charge nicht funktioniert hat, da sie bereits existiert.

In diesem Fall wird der bereits vorhandene Datensatz mit den aktuellen Daten aktualisiert. Dabei werden nur die Felder **ERFMG** (Gewicht), **IEDD** (Enddatum) und **IEDZ** (Endzeitpunkt) überschrieben.

Funktioniert das Aktualisieren, liefert die Funktion **True** zurück, ansonsten wird ein Fehler generiert.

5.2.4 Öffentliche Prozeduren

5.2.4.1 Public Sub Clear()

Parameter: Keine

Initialisiert die Klasse auf den Ausgangszustand. Alle Variablen werden dabei auf Standardwerte gesetzt, wobei Mandant, Werk und TID aus dem globalen Objekt **g_SAP** geholt werden und Datum und Uhrzeit auf den aktuellen Zeitpunkt gestellt werden.

5.2.5 Öffentliche Funktionen

5.2.5.1 Public Function Send() As Boolean

Parameter: ByRef con As ADODB.Connection

Diese Funktion sendet die im Objekt gespeicherten Daten an die Übertragungstabelle in SAP. Dabei werden die Daten so formatiert, dass sie in SAP akzeptiert werden; z.B. werden Datumswerte im Format "**YYYYMMDD**" und Uhrzeiten als "**HHMMSS**" formatiert. Wenn ein Datum oder eine Uhrzeit nicht vorhanden ist, werden entsprechende Nullwerte eingesetzt ("**00000000**" bzw. "**000000**"). Die Auftragsnummer wird mit Nullen links auf 12 Stellen, die Materialnummer auf 18 Stellen aufgefüllt.

Schlägt das Einfügen des Datensatzes fehl, so wird bei den entsprechenden Fehlercodes versucht, einen bereits bestehenden Datensatz mit der Funktion **Update()** zu aktualisieren. Tritt ein anderer Fehlercode auf oder schlägt **Update()** auch fehl, wird ein Fehler erzeugt.

5.3 CSapGlobals

Diese Klasse, von der ein globales Objekt beim Programmstart erzeugt wird, hält die allgemeinen SAP-Informationen zu Mandant, Werk und Transaktions-ID. Weiterhin stellt sie eine Methode zur Verfügung, um Aktionen ans SAP zu signalisieren.

5.3.1 Datentypen und Enums

```
Public Enum SAPAktionConstants
   sapD_AUF_VP        = 1   ' Aufträge von SAP anfordern
   sapU_AUFSTART_VP   = 2   ' wird nicht verwendet
   sapU_AUFENDE_VP    = 3   ' wird nicht verwendet
   sapU_CHARGE_VP     = 4   ' Chargenmeldungen an SAP senden
   sapD_SPEZ_VP       = 5   ' Spezifikationen von SAP anfordern
   sapD_LIEFER_VP     = 6   ' Lieferungsdaten von SAP anfordern
   sapD_BATCH_VP      = 7   ' Chargendaten von SAP anfordern
   sapU_MESSWERT_VP   = 8   ' Messwerte an SAP senden
End Enum
```

Tabelle 19: Enum SAPAktionConstants

Dieser Datentyp dient zur Unterscheidung und Festlegung der Aktionen, die mit der Funktion **SAPAktionSenden()** initiiert werden können.

5.3.2 Interne Variablen / Eigenschaften

Alle Zustände und Daten der Klasse werden in privaten Variablen gehalten. Alle internen Daten werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_sSAPR3_Mandant	**String**	**SAPR3_Mandant**	R/W	Die Mandantennummer, die bei den Vorgängen immer mitgeliefert werden muss. Ist die interne Variable noch nicht gesetzt, wird der Wert aus der globalen Variablen **„g_sSAPMandant“** geholt.
m_sSAPR3_TID	**String**	**SAPR3_TID**	R/W	Die Transaktions-ID, die bei allen Vorgängen angegeben werden muss. Ist die interne Variable noch nicht gesetzt, wird der Wert aus der globalen Variablen **„g_sSAP_TID“** geholt.
m_sSAPR3_Werk	**String**	**SAPR3_Werk**	R/W	Das Kürzel für das Werk, das bei allen Vorgängen angegeben werden muss. Ist die interne Variable noch nicht gesetzt, wird der Wert aus der globalen Variablen **„g_sSAPWerk“** geholt.

Tabelle 20: CSapGlobals, interne Variablen

5.3.3 Interne Funktionen

5.3.3.1 Private Function SAPAktion() As String

Parameter: ByVal Aktion As SAPAktionConstants

Diese Funktion wandelt eine numerische Aktion in einen festgelegten String um, der die entsprechende Aktion kennzeichnet. Diese Strings sind fest definiert und mit dem SAP-Team abgesprochen. Jeder String beginnt mit **U**_ (für Upload) oder **D**_ (für Download) und endet mit **_VP_4B**, wobei **4B** das Werkskürzel darstellt.

Wird ein unbekannter Wert als **Aktion** übergeben, so liefert die Funktion einen Leerstring zurück.

Aktion	*Kommandostring*	*Beschreibung*
sapU_AUFSTART_VP	**U_AUFSTART_VP_4B**	Wird nicht verwendet
sapU_AUFENDE_VP	**U_AUFENDE_VP_4B**	Wird nicht verwendet
sapU_CHARGE_VP	**U_CHARGE_VP_4B**	Chargenmeldungen an SAP senden
sapD_AUF_VP	**D_AUF_VP_4B**	Aufträge von SAP anfordern
sapD_LIEFER_VP	**D_LIEFER_VP_4B**	Lieferdaten von SAP anfordern
sapD_SPEZ_VP	**D_SPEZ_VP_4B**	Spezifikationen von SAP anfordern
sapD_BATCH_VP	**D_BATCH_VP_4B**	Chargendaten von SAP anfordern
sapU_MESSWERT_VP	**U_MESSWERT_VP_4B**	Messwerte an SAP senden

Tabelle 21: CSapGlobals, Aktionen

5.3.4 Öffentliche Funktionen

5.3.4.1 Public Function SAPAktionSenden() As Boolean

Parameter: ByVal Aktion As SAPAktionConstants

Diese Funktion sendet ein Kommando an SAP. Erlaubt sind hierbei als Parameter alle Werte der Enum **SAPAktionConstants**.

Die Aktion wird durch die Funktion **SAPAktion()** in das entsprechende SAP-Kommando übersetzt und zusammen mit Werk, Mandant, TID, sowie Datum und Uhrzeit in die SAP-Tabelle **[ZAKTION_VP_4B]** eingefügt.

Dabei ist zu beachten, dass Sendekommandos erst nach dem Übertragen der eigentlichen Daten geschrieben werden sollten, damit SAP auch die angekündigten Daten finden kann.

5.4 CSapMesswert

Diese Klasse verwaltet einen Messwert in dem Format, wie er zum SAP transportiert wird. Sie bietet Methoden, um die Daten in die SAP-Transfertabellen zu übertragen.

Verwendet wird die Klasse von **CJob** und zwar ausschließlich beim Senden von Messwerten (Funktion **SAP_MesswerteSenden()**). Dort werden jeweils lokale Objekte der Klasse angelegt.

5.4.1 Interne Variablen / Eigenschaften

Alle Zustände und Daten der Klasse werden in privaten Variablen gehalten. Einige davon werden nur intern bearbeitet und sind daher von außen gar nicht sichtbar, andere werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_MANDT	**Long**	**---**	---	Die Mandantennummer zur Anmeldung im SAP. Diese Nummer wird aus dem globalen Objekt **g_SAP** übernommen[27].
m_WERK	**String**	**---**	---	Das Werkskürzel im SAP[28]. Diese Angabe wird aus dem globalen Objekt **g_SAP** übernommen.
m_TID	**String**	**---**	---	Die Transaktions-ID für SAP. Diese Angabe wird aus dem globalen Objekt **g_SAP** übernommen.
m_DATUM	**Date**	**---**	---	Datum und Uhrzeit der Erstellung des Objekts. Wird beim Senden an SAP übertragen.
m_ZEIT	**Date**			
m_State	**String**	**Status**	R/W	Der Status dieses Messwertes. Dabei bedeutet: "**0**" = in Ordnung, "**A**" = SOC-Verletzung.
m_MATNR	**String**	**MATNR**	R/W	Die Materialnummer der Charge, zu der dieser Messwert gehört. Beim Setzen dieses Wertes wird sichergestellt, dass der String genau 18 Zeichen lang ist.
m_CHARG	**String**	**CHARG**	R/W	Die Bezeichnung der Charge, zu der dieser Messwert gehört.
m_VERWMERKM	**String**	**VERWMERKM**	R/W	Das Verwendungsmerkmal dieses Messwertes. Dies ist z.B. **„DP/P“** oder **„SV_BETR“**.
m_MESSWERT	**Double**	**MESSWERT**	R/W	Der eigentliche Messwert.
m_TOLERANZOB	**Double**	**TOLERANZOB**	R/W	Die beiden Toleranzwerte werden nicht verwendet und enthalten immer **0**.
m_TOLERANZUN	**Double**	**TOLERANZUN**	R/W	

Tabelle 22: CSapMesswert, interne Variablen

[27] Hierbei besteht eine Diskrepanz der Datentypen! In **CSapMesswert** wird der Mandant als Long gehalten, in **g_SAP** als String. Da der Inhalt jedoch numerisch ist, stellt das zurzeit kein Problem dar, sollte jedoch im Auge behalten werden.

[28] Dieses lautete für das Werk Offenbach **„4B“**

5.4.2 Interne Prozeduren

5.4.2.1 Private Sub Class_Initialize()

Parameter: Keine

Wird bei der Erstellung der Klasse (z.B. durch **New**) aufgerufen. Ruft die interne Prozedur **Clear()** auf.

5.4.3 Interne Funktionen

5.4.3.1 Private Function Update() As Boolean

Parameter: ByRef con As ADODB.Connection

Wird von der Funktion **Send()** aufgerufen, wenn das Einfügen eines neuen Messwertes nicht funktioniert hat, da er in der Übertragungstabelle bereits existiert.

In diesem Fall wird der bereits vorhandene Datensatz mit den aktuellen Daten aktualisiert. Dabei werden nur die Felder **MESSWERT**, **CHARGE** und **VERWMERKM** (Verwendungsmerkmal) überschrieben.

Funktioniert das Aktualisieren, liefert die Funktion **True** zurück, ansonsten wird ein Fehler generiert.

5.4.4 Öffentliche Prozeduren

5.4.4.1 Public Sub Clear()

Parameter: Keine

Initialisiert die Klasse auf den Ausgangszustand. Alle Variablen werden dabei auf Standardwerte gesetzt, wobei Mandant, Werk und TID aus dem globalen Objekt **g_SAP** geholt werden und Datum und Uhrzeit auf den aktuellen Zeitpunkt gestellt werden.

5.4.5 Öffentliche Funktionen

5.4.5.1 Public Function Send() As Boolean

Parameter: ByRef con As ADODB.Connection

Diese Funktion sendet die im Objekt gespeicherten Daten an die Übertragungstabelle in SAP. Dabei werden die Daten so formatiert, dass sie in SAP akzeptiert werden; z.B. werden Datumswerte im Format "**YYYYMMDD**" und Uhrzeiten als "**HHMMSS**" formatiert. Wenn ein Datum oder eine Uhrzeit nicht vorhanden ist, werden entsprechende Nullwerte eingesetzt ("**00000000**" bzw. "**000000**"). Beim Messwert wird sichergestellt, dass als Dezimaltrennzeichen ein Punkt verwendet wird und der Ergebnisstring nicht länger als 18 Zeichen wird.

Schlägt das Einfügen des Datensatzes fehl, so wird bei den entsprechenden Fehlercodes versucht, einen bereits bestehenden Datensatz mit der Funktion **Update()** zu aktualisieren. Tritt ein anderer Fehlercode auf oder schlägt **Update()** auch fehl, wird ein Fehler erzeugt.

5.5 CSystemStatus

Diese Klasse findet sich in identischer Form in mehreren der PMS-Programme. Sie dient dazu, auf allen Client-Rechnern den Status der Serverprozesse (**PMS_Party** und **PMS_Transfer**) darzustellen.

Für die Überwachung dieser Prozesse wird die Tabelle **[Status]** in der PMS-Datenbank verwendet. Diese Tabelle beinhaltet unter anderem fünf Spalten (**[Status1]**...**[Status5]**), die zur Ermittlung des Status der Verbindungen vorgesehen und folgendermaßen zugeordnet sind:

Feldname	*Zuordnung*	*Beschreibung*
Status1	Status der PLS-Verbindung	Wird von PMS_Party mit verwaltet. Diese Spalte meldet, ob die Verbindung zum PLS hergestellt werden konnte.
Status2	Status der SAP-Verbindung	Wird von PMS_Transfer mit verwaltet. Diese Spalte meldet, ob die Verbindung zum SAP hergestellt werden konnte.
Status3	noch nicht zugeordnet	noch keine Bedeutung
Status4	noch nicht zugeordnet	noch keine Bedeutung
Status5	noch nicht zugeordnet	noch keine Bedeutung

Prinzipiell funktioniert der Mechanismus wie ein Totmannschalter: der Prozess (hier natürlich **PMS_Party**) muss in bestimmten Abständen (maximal 5 Minuten) seinen Datensatz in der Tabelle aktualisieren (unter anderem auch mit der aktuellen Uhrzeit), damit der Eintrag nicht veraltet. Die Client-Prozesse (z.B. **PMS_Rezeptur** und **PMS_Stammdaten**) kontrollieren — auch durch eine Instanz der Klasse **CSystemStatus** —, ob a) ein aktiver Eintrag vorhanden und b) der Eintrag noch gültig ist. Treffen beide Bedingungen zu, so gilt der überwachte Prozess als funktionsfähig. Anderenfalls wird in den Clients die Anzeige (als grüne, gelbe und rote Lampen dargestellt) auf Fehler gestellt und somit der Benutzer darüber informiert, dass ein Problem vorliegt.

Grün bedeutet dabei, dass der überwachte Prozess in Ordnung ist.

Gelb zeigt an, dass der Status nicht ermittelt werden konnte. Dies kann vorkommen, wenn zum Beispiel der SAP-Transfer nicht läuft. Da dieses Programm aber auch für den Status der Verbindung zum SAP verantwortlich ist, wird die „SAP-Lampe“ in diesem Fall auf gelb gestellt.

Rot heißt, dass der Prozess entweder gar nicht läuft oder seit über 5 Minuten seinen Datensatz nicht mehr aktualisiert hat. In beiden Fällen gilt der Prozess als „verstorben“ und eine Kontrolle wird notwendig.

Im Gegensatz zu den meisten anderen Klassen im Programm hält diese Klasse eine komplett eigenständige Verbindung zur Datenbank, um so unabhängig wie möglich zu sein.

5.5.1 Datentypen und Enums

```
Public Enum stStatusType
   stStatusUnknown = 0
   stStatusPLS     = 1
   stStatusSAP     = 2
   stStatus3       = 3
   stStatus4       = 4
   stStatus5       = 5
   stStatusMax     = 5
End Enum
```

Tabelle 23: Enum stStatusType

Dieser Datentyp listet die verschiedenen Status auf, die in der Klasse verarbeitet werden können.

```
Private Type tyConn
   sConnString     As String
   vCurLocation    As Variant
   nConnTimeout    As Long
   nCmdTimeout     As Long
End Type
```

Tabelle 24: Datentyp tyConn

Dieser Datentyp enthält die notwendigen Informationen zur Verbindung mit der Datenbank. Beim Verbinden mit der DB werden die entsprechenden Angaben aus der übergebenen Verbindung in eine interne Struktur dieses Typs übernommen und dann daraus eine neue Verbindung zur DB hergestellt.

5.5.2 Interne Konstanten

CSystemStatus ist die einzige Klasse, die ihre SQL-Statements selbst enthält. Der Grund dafür ist der, dass die Klasse in verschiedenen Programmen (**PMS_Stammdaten**, **PMS_Rezeptur**, **PMS_Party**, **PMS_Transfer**, etc.) verwendet wird[29], die SQL-Kommandos aber immer dieselben sind. Daher wäre es Unsinn, diese Verbindung mit Gewalt zu zerreißen.

Die Funktion **SQLFormat()** wird jedoch extern erwartet (ist hier in **basSQL.bas** vorhanden).

5.5.2.1 sSqlP_Check

Dieses Statement dient nur zum Prüfen, ob die Datenbank ansprechbar ist. Es benötigt keine Tabelle und eigentlich auch keine Datenbank zur Ausführung.

5.5.2.2 sSqlP_LoadStatus

Lädt den letzten aktiven Status des angegebenen Programms aus der DB.

5.5.2.3 sSqlP_LoadStatusByID

Wie **sSqlP_LoadStatus**, nur dass hier anstelle des Programmnamens die ID verwendet wird.

[29] Die Verbindung der Quellcodes wird durch Microsoft® Visual SourceSafe® sichergestellt.

5.5.2.4 sSqlP_UpdStatus

Mit diesem Kommando wird der Status des Programms im Modus „Lebendig“ gehalten, indem hauptsächlich das letzte Änderungsdatum und der Statuswert aktualisiert werden.

5.5.2.5 sSqlP_InsStatus

Dieses Kommando fügt den Status des Programms ein, falls er noch nicht vorhanden ist. Genau wie **`sSqlP_UpdStatus`** wird es in der Funktion **`SaveStatus()`** verwendet.

5.5.2.6 sSqlP_StopStatus

Dieses Kommando schließt den aktuellen Datensatz ab, indem das Feld **`[Ende]`** auf die aktuelle Uhrzeit gesetzt wird.

5.5.2.7 sSqlP_GetStatus

Prüft, ob ein bestimmter Status in Ordnung ist. „In Ordnung“ bedeutet dabei, dass ein „offener“ Datensatz mit diesem Status existiert, der Status nicht **`NULL`** ist und der Datensatz nicht veraltet ist.

5.5.2.8 sSqlP_GetProgStatus

Ähnlich wie **`sSqlP_GetStatus`**, nur wird hierbei der Status eines Programms geprüft.

5.5.3 Interne Variablen / Eigenschaften (Properties)

Alle Zustände und Daten der Klasse werden in privaten Variablen gehalten. Einige davon werden nur intern bearbeitet und sind daher von außen gar nicht sichtbar, andere werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_Con	**ADODB.Connection**	**Connection**	W	Die interne Datenbankverbindung
m_ProgName	**String**	**ProgName**	R/W	Der Name des Programms, das hier überwacht wird.
m_ID	**Long**			Die interne Datensatz-ID, wird von der Datenbank geliefert.
m_Status	**stStatusType**	**StatusType**	R/W	Der Typ des zu speichernden Status – der Inhalt ist abhängig von **`m_OK`**.
m_OK	**Boolean**	**Status**	R/W	**`True`**, wenn das Programm oder die Verbindung in Ordnung ist, sonst **`False`**.
m_MaxAge	**Long**	**MaxAge**	R/W	Das maximale Alter des Status in Sekunden. (Default = 300)
m_ConnInfo	**tyConn**	**Connection**	W	die Verbindungsinformationen

Tabelle 25: CSystemStatus, Interne Variablen und Eigenschaften

5.5.4 Interne Prozeduren

5.5.4.1 Private Sub Class_Initialize()

Parameter: Keine

Wird bei der Erstellung der Klasse (z.B. durch **New**) aufgerufen. Stellt die internen Variablen auf Standardwerte.

5.5.4.2 Private Sub Class_Terminate()

Parameter: Keine

Wird beim Zerstören der Klasse (z.B. durch **Set xxx = Nothing**) aufgerufen. Schließt die DB-Verbindung und zerstört das zugehörige Objekt **m_Con**.

5.5.5 Interne Funktionen

5.5.5.1 Private Function CheckConn() As Boolean

Parameter: Keine

Prüft mit dem Kommando **sSqlP_Check**, ob die Datenbank erreichbar ist. Wenn ja, liefert die Funktion sofort **True**. Wenn nicht, wird versucht, eine neue Verbindung herzustellen. Schlägt dies fehl, wird **m_Con** auf **Nothing** gestellt und die Funktion liefert **False**.

5.5.6 Öffentliche Funktionen

5.5.6.1 Public Function Load() As Boolean

Parameter: Keine

Lädt den letzten Status dieses Programms aus der Datenbank. Dabei wird der Status analysiert und die internen Variablen eingeschaltet,

5.5.6.2 Public Function SaveStatus() As Boolean

Parameter: Keine

Speichert den aktuellen Status dieses Programms in der Datenbank. Sollte noch kein Datensatz existieren, so wird er dabei angelegt, anderenfalls nur aktualisiert.

5.5.6.3 Public Function StopStatus() As Boolean

Parameter: Keine

Diese Funktion beendet die Überwachungsfunktion der Klasse. Sie speichert die aktuelle Uhrzeit im Datenbankfeld **[Ende]**, wodurch das Programm als „beendet“ gilt.

5.5.6.4 Public Function CheckState() As Boolean

Parameter: ByVal stat As stStatusType

Prüft, ob der angegebene Status in der Datenbank in Ordnung ist. Dazu muss das entsprechende Feld **[StatusX]** einen Wert ungleich **NULL** enthalten, das Feld **[Ende]** muss dagegen **NULL** sein und **[LastChange]** plus **[MaxAge]** darf nicht kleiner als das aktuelle Datum sein. Trifft dies alles zu, liefert die Funktion **True**, sonst **False**.

5.5.6.5 Public Function CheckProg() As Boolean

Parameter: ByVal sProg As String
ByRef sWorkstation As String

Diese Funktion prüft, ob das Programm im Parameter **sProg** aktuell läuft. Wenn ja, liefert sie True und trägt den Rechnernamen des Datensatzes im Parameter **sWorkstation** ein. Wenn nicht, liefert sie **False** und leert **sWorkstation**. Sollte die Verbindung zur Datenbank nicht hergestellt werden können, wird ebenfalls **False** geliefert, jedoch enthält **sWorkstation** in diesem Fall den Text **„#unknown#“**.

5.6 CSysTray

Diese Klasse übernimmt das Minimieren des Programms in den Traybereich der Taskleiste von Windows. Anstatt das Fenster wie gewöhnlich zu minimieren, wird ein Symbol im Traybereich erzeugt und die normale minimierte Darstellung im Programmbereich der Taskleiste unterdrückt.

Weiterhin wird das Klicken auf dem Traysymbol behandelt, wodurch ein Menü angezeigt oder eine bestimmte Reaktion (z.B. das Fenster wieder sichtbar zu machen) hervorgerufen werden kann.

Auch diese Klasse wird in identischer Form von mehreren Programmen (z.B. **PMS_Transfer**, **PMS_Party** und **PMS_DbAlarm**) verwendet.

Da der Zugriff auf den Traybereich selbstverständlich nicht von Basic-Methoden aus geschehen kann, musste hier auch wieder auf Systemfunktionen zurückgegriffen werden.

5.6.1 Datentypen und Enums

```
Private Type NOTIFYICONDATA
   cbSize             As Long
   hwnd               As Long
   uId                As Long
   uFlags             As Long
   ucallbackMessage   As Long
   hIcon              As Long
   szTip              As String * 64
End Type
```

Tabelle 26: Datentyp NOTIFYICONDATA

Dieser Datentyp wird für die verschiedenen Systemfunktionen der Klasse benötigt.

5.6.2 Interne Konstanten

Die hier verwendeten Konstanten sind Long-Definitionen, die zum Aufruf der entsprechenden Funktionalitäten der Systemfunktionen sowie zum Erzeugen der Ereignisse benötigt werden.

Konstante	*Wert*	*Beschreibung*
NIM_ADD	**&H0000**	Fügt ein neues Symbol im Traybereich ein.
NIM_MODIFY	**&H0001**	Modifiziert das Verhalten des Symbols im Traybereich.
NIM_DELETE	**&H0002**	Löscht das Symbol aus dem Traybereich
NIF_MESSAGE	**&H0001**	Flag: Bewirkt, dass Nachrichten vom Symbol an das Programm weitergeleitet werden.
NIF_ICON	**&H0002**	Flag: Bewirkt, dass bei der Initialisierung das Icon angegeben werden kann.
NIF_TIP	**&H0004**	Flag: Bewirkt, dass bei der Initialisierung ein Tooltip angegeben werden kann.
WM_MOUSEMOVE	**&H0200**	Windows-Nachricht „Maus wurde bewegt“.
WM_LBUTTONDBLCLK	**&H0203**	Windows-Nachricht „Doppelklick links“
WM_LBUTTONDOWN	**&H0201**	Windows-Nachricht „Linker Mausklick“
WM_LBUTTONUP	**&H0202**	Windows-Nachricht „Linke Maustaste losgelassen“
WM_RBUTTONDBLCLK	**&H0206**	Windows-Nachricht „Doppelklick rechts“
WM_RBUTTONDOWN	**&H0204**	Windows-Nachricht „Rechter Mausklick“
WM_RBUTTONUP	**&H0205**	Windows-Nachricht „Rechte Maustaste losgelassen“

Tabelle 27: CSysTray, Konstanten

5.6.3 Interne API-Deklarationen

5.6.3.1 Declare Function Shell_NotifyIcon() As Boolean

Lib: *shell32*

Alias: *Shell_NotifyIconA*

Parameter: *ByVal dwMessage As Long*
pnid As NOTIFYICONDATA

Diese Methode wird zur Kommunikation und Einstellung des Icons im Traybereich benötigt. Der Parameter **`pnid`** muss vor dem Aufruf entsprechend gefüllt werden.

5.6.4 Interne Variablen / Eigenschaften (Properties)

Die Zustände und Daten, bei denen es möglich ist, werden in privaten Variablen gehalten. Einige davon werden nur intern bearbeitet und sind daher von außen gar nicht sichtbar, andere werden durch Eigenschaften (Properties) zugänglich gemacht.

Variablenname	*Typ*	*Property*	*R/W*	*Beschreibung*
m_Notify	**NOTIFYICONDATA**			Struktur, die zur Kommunikation mit dem System benötigt wird
m_PicHook	**PictureBox**			Pseudofenster, das u.a. benötigt wird, um Ereignisse auszulösen.
m_sToolTip	**String**	**ToolTip**	R/W	Text, der als Tooltip angezeigt wird, sobald der Mauszeiger über dem Icon liegt.
m_frmSrcWnd	**Form**	**SourceWindow**	R/W	Fenster, das von der Klasse beeinflusst wird; normalerweise das Fenster, das auch die Klasse erzeugt.
m_bDefDblClk	**Boolean**	**DefaultDblClk**	R/W	**`True`**, wenn ein Doppelklick das Standardverhalten bewirken soll (Wiederherstellen des Fensters). **`False`**, wenn nur ein normales Ereignis gesendet werden soll.
m_bVisible	**Boolean**			Flag, das den aktuellen Zustand (sichtbar/unsichtbar) des Icons beinhaltet.

Tabelle 28: CSysTray, Interne Variablen und Eigenschaften

5.6.5 Interne Prozeduren

5.6.5.1 Private Sub Class_Initialize()

Parameter: Keine

Wird bei der Erstellung der Klasse (z.B. durch **`New`**) aufgerufen. Stellt die internen Variablen auf Standardwerte.

5.6.5.2 Private Sub Class_Terminate()

Parameter: Keine

Wird beim Zerstören der Klasse (z.B. durch **`Set xxx = Nothing`**) aufgerufen. Wenn das Symbol sichtbar ist, wird es aus dem Traybereich entfernt.

5.6.5.3 Private Sub m_PicHook_MouseMove()

Parameter: *Button As Integer*
Shift As Integer
x As Single
y As Single

Diese Methode wird als Callback aufgerufen, wenn eine Aktion notwendig wird. Folgende Aktionen werden unterstützt:

Doppelklick links	Ruft die interne Methode **`LButtonDblClk()`** auf.
Linker Button gedrückt	Erzeugt das Ereignis **`LButtonDown`**.
Linker Button losgelassen	Erzeugt das Ereignis **`LButtonUp`**.
Doppelklick rechts	Erzeugt das Ereignis **`RButtonDblClk`**.
Rechter Button gedrückt	Erzeugt das Ereignis **`RButtonDown`**.
Rechter Button losgelassen	Erzeugt das Ereignis **`RButtonUp`**.

5.6.5.4 Private Sub LButtonDblClk()

Parameter: *Keine*

Wenn **`m_bDefDblClk`** **`True`** ist, wird die Methode **`RestoreFromSysTray()`** aufgerufen. Danach wird immer das Ereignis **`LButtonDblClk`** erzeugt.

5.6.5.5 Private Sub SetPicHook()

Parameter: *Keine*

Erstellt ein unsichtbares Fenster vom Typ **`PictureBox`** im Elternfenster. Wird von der Eigenschaft **`SourceWindow`** (Set) aufgerufen. Dieses neue Fenster wird benötigt, um z.B. dem Elternfenster Nachrichten zu senden.

5.6.6 Öffentliche Prozeduren

5.6.6.1 Public Sub RemoveFromSysTray()

Parameter: *Keine*

Entfernt das Icon aus dem Systray. Wenn das Icon noch nicht oder nicht mehr angezeigt ist, passiert nichts (gesteuert durch **`m_bVisible`**).

5.6.6.2 Public Sub IconInSysTray()

Parameter: *Keine*

Zeigt das Icon im Systray an. Ist das Icon bereits sichtbar, so wird es zuerst entfernt. Dies kann dazu verwendet werden, das Icon an den linken Rand des Systrays zu verschieben (wozu auch immer).

5.6.6.3 Public Sub MinToSysTray()

Parameter: Keine

Minimiert das Elternfenster, macht es in der normalen Taskleiste unsichtbar und zeigt das Icon im Systray an.

5.6.6.4 Public Sub RestoreFromSysTray()

Parameter: Keine

Normalisiert das Elternfenster und macht es sowohl als Fenster als auch in der normalen Taskleiste sichtbar.

Weiterhin wird noch kontrolliert, ob das Fenster außerhalb des Bildschirms liegt. Wenn ja, wird es an die Position 10,10 gesetzt (Links oben im Bildschirm).

5.6.7 Ereignisse (Events)

Events werden erzeugt (im Basic-Sprachgebrauch: gefeuert), wenn bestimmte Ereignisse eingetroffen sind. Diese Ereignisse können dann in dem Fenster, das ein Objekt der Klasse **CSysTray** erstellt hat, empfangen werden. Dazu ist es jedoch wichtig, das Objekt mit dem Attribut **WithEvents** zu versehen. Beispiel:

```
Private WithEvents m_SysTray As CSysTray
```

Tabelle 29: Attribut WithEvents

Diese Zeile (außerhalb der Prozeduren) definiert ein Objekt namens **m_SysTray**, das durch das Attribut **WithEvents** auch Events feuern kann, die dann durch entsprechende Methoden (z.B. **m_SysTray_LButtonDblClk()**) bearbeitet werden können.

Hinweis: Auch wenn Events Parameter mitliefern können, so sind doch alle hier verwendeten ohne Parameter.

5.6.7.1 LButtonDblClk

Wird gefeuert, wenn mit der linken Maustaste doppelt auf das Symbol geklickt wurde. Wenn die Eigenschaft **DefaultDblClk** eingeschaltet ist, wird vor dem Feuern dieses Events noch als Standardbehandlung die Prozedur **RestoreFromSysTray()** ausgeführt (Diese Sonderbehandlung geschieht ausschließlich bei **LButtonDblClk**!)

5.6.7.2 LButtonDown

Wird gefeuert, wenn die linke Maustaste auf dem Symbol gedrückt wird.

5.6.7.3 LButtonUp

Wird gefeuert, wenn die linke Maustaste auf dem Symbol losgelassen wird.

5.6.7.4 RButtonDblClk

Wird gefeuert, wenn mit der rechten Maustaste doppelt auf das Symbol geklickt wurde.

5.6.7.5 RButtonDown

Wird gefeuert, wenn die rechte Maustaste auf dem Symbol gedrückt wird.

5.6.7.6 RButtonUp

Wird gefeuert, wenn die rechte Maustaste auf dem Symbol losgelassen wird. Dieses Ereignis eignet sich optimal dafür, ein Menü anzuzeigen, in dem verschiedene Standardfunktionen angeboten werden können.

6 Zusatzprogramme

6.1 TEcho

TECHO.EXE ist ein Kommandozeilenprogramm und dient dazu, Texte sowohl auf der Konsole auszugeben, als auch in eine Datei zu schreiben. Es eignet sich gut dafür, wiederkehrende Prozesse zu protokollieren.

6.1.1 Parameter

Durch den Parameter **-?** wird der folgende Text angezeigt:

```
TECHO - Textausgabe mit Datum und Uhrzeit
        (c) 2002 by: RaisE Raithel Service & Entwicklung

Syntax: TECHO [-?][-D][-T][-F<Datei>] [beliebiger Text]
  -? zeigt diese Hilfe an
  -D unterdrückt die Datumsanzeige
  -T unterdrückt die Zeitanzeige
  -F hängt die Ausgabe auch an <Datei> an

Der Text kann die folgenden Sonderzeichen enthalten,
die sonst nicht darstellbar wären:
  $A  - & (kaufmännisches Und)
  $B  - | (Pipe, Verkettung)
  $E  - Escape (ASCII-Code 27)
  $G  - > (Grösser-als)
  $L  - < (Kleiner-als)
  $H  - Backspace
  $_  - Zeilenumbruch

Alle anderen Zeichen werden einfach ausgegeben.
```

6.1.2 Beispiele:

```
C:\>techo Hallo
2003-12-03 11:37:07> Hallo

C:\>techo -d Hallo
11:37:09> Hallo

C:\>techo -t Hallo
2003-12-03> Hallo
```

Zur Beachtung: Um ein doppeltes Hochkomma (") zu erzeugen, müssen Sie DREI angeben!

Beispiele:

```
C:\>techo "Hallo"
2003-12-03 11:41:12> Hallo

C:\>techo ""Hallo""
2003-12-03 11:41:14> Hallo

C:\>techo """Hallo"""
2003-12-03 11:41:16> "Hallo"
```

6.2 WaitFor

WAITFOR.EXE prüft, ob ein Fenster mit einem frei wählbaren Titel existiert und wartet einfach darauf, dass dieses Fenster wieder verschwindet. Auf diese Weise lassen sich Batchabläufe recht einfach mit normalen, fensterbasierten Programmen synchronisieren.

6.2.1 Parameter

Wird **WaitFor** ohne Parameter aufgerufen, so wird das folgende Fenster angezeigt:

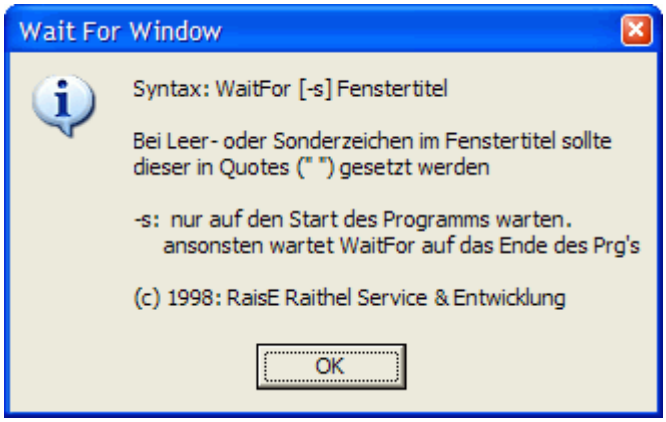

Abbildung 6: WaitFor, Hilfefenster

Es gibt nur einen Parameter für das Programm, nämlich den Titel des Fensters, auf das **WaitFor** warten soll. Wenn der Titel Leer- oder Sonderzeichen enthält, so empfiehlt es sich, diesen in Anführungszeichen (**"Der Titel"**) zu setzen.

Durch die Option **-s** wartet **WaitFor** nicht auf das Ende des gesuchten Programms, sondern nur darauf, dass es gestartet wird und beendet sich dann.

Um nicht allzu viel Rechnerzeit zu verschwenden, prüft **WaitFor** nur alle 0,5 Sekunden, ob das Fenster erscheint, bzw. wieder verschwunden ist.

6.2.2 Beispiel:

```
1: @echo off
2: title Keep Transfer
3:
4: :Loop
5: start /min PMS_Transfer.exe
6: techo -fPMS_Logging.txt SAP-Transfer wurde gestartet
7: WaitFor "PMS SAP-Transfer"
8: techo -fPMS_Logging.txt SAP-Transfer wurde beendet!
9: goto Loop
```

Tabelle 30: KeepAlive-Batch

Die dargestellte CMD-Datei sorgt dafür, dass das Programm **PMS_Transfer** auf jeden Fall läuft. Dazu wird das Programm in Zeile 5 über das Kommando **START** gestartet und in Zeile 6 ein Eintrag in der Logdatei vorgenommen. In Zeile 7 wird **WaitFor** aufgerufen, das nun zuerst darauf wartet, dass das Fenster mit dem Titel „PMS SAP-Transfer“ erscheint. Ist das geschehen, wartet es nun darauf, dass dieses Fenster wieder verschwindet. Zu diesem Zeitpunkt beendet sich **WaitFor** und die CMD-Datei läuft weiter.