PMS - Trends einstellen und anzeigen

Technische Beschreibung von PMS_Trend und PMS_Display

erstellt von Klaus Raithel
Version 87 (04.09.06)
(in Bearbeitung)

INVISTA

COMPUTER OUTFIT

Inhalt

Tabellen

Abbildungen

Codebeispiele

1 Allgemeines

Bei Invista werden Anlagedaten der Betriebe P2K, P2D, DMT, FSK und P3K mittels eines Datenerfassungssystems der Firma WonderWare erfasst. Dieses Erfassungssystem ist pro Betrieb auf jeweils zwei redundant ausgelegten Servern installiert. Diese Systeme erfassen zyklisch die von den Anlagen über verschiedene Geräte (z.B. Erdmann PJ40) gelieferten Daten und speichern sie auf dem jeweiligen Server in eigenen Dateien.

Diese Dateien sind sowohl vom ersten, als auch dem zweiten Server des Betriebes erreichbar. Darüber wird der Stand der beiden Systeme derart abgeglichen, dass beide über den gleichen Datenbestand verfügen. Beim Ausfall eines Servers kann der jeweils andere die Datenerfassung weiterführen.

Ein weiterer wichtiger Punkt ist, dass die Daten über einen Microsoft® SQL Server bereitgestellt werden, wodurch die Daten mit Standardverfahren in allen Programmen verwendet werden können.

Außerdem stellt WonderWare ein ActiveX-Control zur Verfügung, mit dem ein oder mehrere Messstellen in grafischer Weise als Trendkurven dargestellt werden können. Dieses Control namens **iTrend.ocx** wird in den hier vorgestellten Programmen **PMS_Trend** und **PMS_Display** verwendet, um die Kurven am Bildschirm darzustellen, bzw. zu bearbeiten.

1.1 Aufgabe von PMS_Trend

PMS_Trend dient dazu, die Kurvensätze, die später in **PMS_Display** dargestellt werden sollen, zu definieren. Dazu wird – mittels **iTrend** – ein Kurvensatz erstellt, im Fenster dargestellt und bearbeitet.

1.2 Aufgabe vom PMS_Display

PMS_Display dient zum einen dazu, ein bis acht der in **PMS_Trend** erstellten Kurvensätze in einer speziellen Datei mit der Endung **.CRVL** zusammenzufügen und zum zweiten diese so kombinierten Kurvensätze dann auf dem Bildschirm darzustellen. Ebenfalls in **PMS_Display** können die eigentlichen Diagramme dann live dargestellt werden. Dabei können auf dem ersten und auf dem zweiten Monitor jeweils bis zu vier Diagramme dargestellt werden.

1.3 Dateiformate

PMS_Trend verwendet kein eigenes Dateiformat, sondern lädt und speichert die Daten im Format von **iTrend**, das Dateien mit der Endung **.CRV** verwendet. Dies sind reine Textdateien, die wie eine INI-Datei aufgebaut sind und einen kompletten Kurvensatz beschreiben.

2 PMS_Trend

2.1 Externe Module

Wie jedes VB-Programm benötigt auch PMS_Trend einige externe Module in Form von Referenzen und Komponenten.

2.1.1 Referenzen

Name	Zweck
Visual Basic runtime objects and procedures	Diese Elemente werden für die grundlegenden Teile der VB-Programmierung benötigt
Visual Basic objects and procedures	
Visual Basic for Applications	
MS ActiveX Data Objects 2.7 Library	Stellt Objekte und Methoden für den Datenbankzugriff zur Verfügung
SSLib UI VB Library 2.1	Wird nur für den Info-Dialog benötigt.
SSLib Universal VB Library 2.1	Liefert einige interessante Objekte und Methoden, hauptsächlich für den Zugriff auf die Registrierung.
Microsoft CDO For Exchange 2000 Library	Wird für das Versenden von Mails im Falle eines Programmfehlers benötigt.
MS Scripting Runtime	Bringt Objekte, mit denen der Dateizugriff optimiert werden kann.
PMS_Mod	Schnittstelle zu WonderWare
ActiveFactory UI support objects package (8.5)	Liefert alle Komponenten zur Anzeige der Kurven

Tabelle 1: Referenzen

2.1.2 Komponenten

Name	Zweck
MS Common Dialog Control 6.0	Allgemeine Dialoge, wie Datei öffnen und speichern, Drucken oder Farbauswahl.
MS Windows Common Controls 6.0 (SP6)	Stellt Windows-Elemente wie Listcontrol, TreeView, etc. zur Verfügung.

Tabelle 2: Komponenten

2.2 Quelldateien – Übersicht

PMS_Trend besteht – abgesehen von den externen Modulen – insgesamt aus 5 Basic-Modulen, 2 Klassen und 7 Fenstern.

2.2.1 Basic-Module

Dateiname	Zweck
basMain.bas	Allgemeine Methoden; Startprozedur „`Main`“.
basError.bas	Fehlerbehandlungen.
basSQL.bas	SQL-Kommandostrings, sowie verschiedene Methoden zum Zugriff auf Datenbanken.
MonitorInfo.bas	Methoden zum Ermitteln der Monitorinformationen und zum Ansprechen der verschiedenen Bildschirme.
MultiHook.bas	Liefert Methoden zur Überwachung der Windows-Nachrichten; wird hier dazu verwendet, um eine minimale Fenstergröße festlegen zu können.

Tabelle 3: Basic-Module

2.2.2 Klassen

Klassenname	Dateiname	Zweck
CSecurity	CSecurity.cls	Methoden zur Anmeldung am PMS-System.
CSplitter	CSplitter.cls	Stellt eine Trennleiste zur dynamischen Größenänderung von Teilfenstern zur Verfügung.

Tabelle 4: Klassen

2.2.3 Fenster und Dialoge

Fenstername	Dateiname[1]	Zweck
frmTrend	frmTrend.frm	Das Hauptfenster.
dlgAbout	dlgAbout.frm	Dialog mit allgemeinen Angaben über Programmversion und die eingebundenen Module.
dlgErrHndl	dlgErrHndl.frm	Dialog zur Anzeige von Fehlermeldungen mit der Option, diese per Mail zu versenden.
dlgGetUser	dlgGetUser.frm	Dialog zur Freigabe von Funktionen, die eine höhere Authentifizierung benötigen, als der Benutzer hat.
dlgLogin	dlgLogin.frm	Dialog zur Anmeldung am PMS-System; dient auch als „Splashscreen"
dlgOptions	dlgOptions.frm	Optionsdialog zur Eingabe der Anmeldeinformationen an der PMS-Datenbank[2].
dlgSaveAs	dlgSaveAs.frm	Dialog zum Speichern der Kurvensätze in den erlaubten Verzeichnissen.

Tabelle 5: Fenster

[1] Zu jeder `.frm`-Datei gehört auch noch jeweils eine `.frx`-Datei, die binäre Ressourcen, wie Bilder und Icons enthält.

[2] Das hat nichts mit der Anmeldung am PMS-System zu tun; diese geschieht später und wird mit Infos aus der Datenbank verifiziert.

2.3 Basicmodule – Detail

2.3.1 basMain

Diese Datei enthält vor allem das `Sub Main()`, das beim Programmstart ausgeführt wird und alle weitere Vorgänge initiiert.

Weiterhin sind hier globale Variablen, Typen, Enumerationen und externe Funktionen deklariert.

2.3.1.1 Datenelemente

Name	Typ	Zweck
g_Security	CSecurity	Security-Objekt für den aktuellen Benutzer zur Prüfung der Zugriffsberechtigung
g_FormLoaded	Boolean	Flag, das gesetzt wird, nachdem das Hauptfenster initialisiert ist
g_StartPos	Integer	die Bildschirmposition, an der das Fenster angezeigt werden soll (0 = Default)
g_sStartFile	String	Datei, die beim Start geladen werden soll (relativ zum Datenverzeichnis „`iTrends`“)
g_bHideStatus g_bHideMenu	Boolean	Flags, die angeben, ob die Status- oder die Menüzeile unterdrückt werden soll (können per Option `/H:S`, bzw. `/H:M` gesetzt werden)
g_sDataServer g_sDataUsr g_sDataPwd g_sShareDrive g_sHomeDir	String	der Server, von dem die Daten und Trendfiles gelesen werden, sowie die Login-Infos, das zu verwendende Laufwerk und das Benutzerverzeichnis relativ zum Datenverzeichnis „`iTrends`“.
csOptPlant csOptUser csOptPwd csOptLocation csOptFile csOptHide	Const String	die möglichen Optionsschalter; als Startzeichen sind hier sowohl „`/`“, als auch „`-`“ erlaubt.
csWWUserName csWWUserPwd	Const String	Benutzername und Kennwort zur Anmeldung an den Wonderware-Servern. Sollten diese irgendwann geändert werden, müssen auch diese Konstanten angepasst werden.
g_chRegRoot g_csRegPath g_csWinLeft g_csWinTop g_csWinWidth g_csWinHeight g_csWinState g_csRootPath g_csTreePos	Const String	Registry-Keys[3] für persistente Einstellungen unter `HKEY_LOCAL_MACHINE\Software\KoSa\PMS\Trend` Fensterposition Fenstergröße Fensterstatus (Normal, Maximiert, etc.) das Startverzeichnis des Betriebes die Position des Splitters
g_csTagPicker g_csToolbars g_csTagDetails g_csInterpolation	Const String	Registry-Keys für verschiedene `iTrend`-Flags: Tagliste sichtbar / Toolbars sichtbar / Tagdetails sichtbar / Interpolationsmodus (Kurvenglättung)

[3] diese werden in der lokalen Registry unter `HKEY_LOCAL_MACHINE` gespeichert und sind daher für **alle** Benutzer gleich.

csDirITrends csDirBasis csDirUser csDirUserA csDirUserB csDirUserC csDirUserD csDirUserMsr	Const String	Verschiedene Verzeichnisse unterhalb dem Verzeichnis **iTrends** auf dem Datenserver
g_csRegDbPath g_csRegDbServer g_csRegDbUser g_csRegDbPwd g_csRegDbCatalog g_csRegDbLocalServer	Const String	Registry-Keys für persistente Einstellungen zur PMS-Datenbank unter **HKEY_LOCAL_MACHINE\Software\KoSa\PMS\SQLDB**
eSecurity	Enum	Enumeration der verschiedenen Sicherheitsebenen für den aktuellen Benutzer
STARTUPINFO PROCESS_INFORMATION	Type	Datenstrukturen für die Windows-Funktionen zum Starten externer Programme
NORMAL_PRIORITY_CLASS INFINITE STARTF_USESHOWWINDOW SW_HIDE	Const Long	Konstanten zum Aufruf der Windows-Funktionen zum Starten externer Programme
CreateProcess WaitForSingleObject CloseHandle GetExitCodeProcess	Function	Windows-Funktionen um externe Programme synchron starten zu können (werden hier zum Aufruf von **NET.EXE** zur Authentifizierung verwendet)
DRIVE_UNKNOWN DRIVE_NO_ROOT_DIR	Const Long	Konstanten zum Aufruf von **GetDriveType**
GetDriveType	Function	Windows-Funktion zum Ermitteln des Laufwerkstyps

Tabelle 6: basMain, Datenelemente

2.3.1.2 Funktionen und Methoden

2.3.1.2.1 Public Sub Main()

Parameter: —

Rückgabe: —

Diese Methode wird beim Programmstart aufgerufen. Hier wird als erstes mit der Funktion **GetFreeDrive()** das letzte unbenutzte Laufwerk ermittelt[4], das später zur Verbindung mit dem Datenserver verwendet wird.

Als nächstes wird in einer Schleife versucht, die Verbindung zur PMS-Datenbank herzustellen. Sind in der Registry keine oder ungültige Anmeldewerte vorhanden, wird **dlgOptions** angezeigt, wo der Benutzer gültige Werte für die Datenbank angeben muss. Die Schleife endet dann, wenn gültige Werte zur Verfügung stehen oder der Benutzer die Eingabe abbricht. In diesem Fall wird das Programm sofort beendet.

Nun wird das globale Objekt **g_Security** erstellt, das bei der Erstellung alle Anmeldeeigenschaften, Berechtigungen, etc. für den Benutzer lädt.

[4] normalerweise ist das Laufwerk **Z:**, ist dieses bereits belegt, dann **Y:**, usw.

Der nächste Schritt liest die Kommandozeile mit den Optionen und Parametern. Folgende Optionen werden erkannt:

Option	Parameter	Zweck
`/PLANT:`	Betriebskürzel	Legt fest, an welchem Betrieb sich das Programm anmeldet. Möglich sind dabei: **P2D**, **P2K**, **P3K**, **DMT** oder **FSK**.
`/USR:`	Benutzername des PMS-Systems	Dies ist der Name eines Benutzers, wie er im PMS festgelegt ist.
`/PWD:`	Kennwort zum Benutzer	Das Kennwort zum Benutzer. Hat der Benutzer kein Kennwort, kann diese Option einfach weggelassen werden.
`/L:`	Fensterposition	Hier kann die Position des Fensters festgelegt werden. Möglich sind dabei Werte von 1 … 8[5].
`/F:`	Zu ladende Datei	Hier kann eine `.CRV`-Datei angegeben werden, die dann automatisch geladen und aktiviert wird.
`/H:`	Unterdrücken von Fensterelementen	Mit `/H:S` wird die Statuszeile unterdrückt, mit `/H:M` die Menüleiste. Die beiden Optionen könen auch kombiniert werden: `/H:SM` oder `/H:MS`.

Tabelle 7: Kommandozeilen-Optionen

Bei den Optionen gilt (außer beim Kennwort!) prinzipiell, dass die Schreibweise unerheblich ist. Der Benutzername „`/usr:P2KUSER`" funktioniert genau so gut wie z. B. „`/Usr:P2KUser`" oder „`/USR:p2kuser`". Weiterhin müssen die Parameter direkt an den Doppelpunkt angehängt werden (also „`/plant:P2K`" und nicht „`/plant : P2K`"!). Bei der zu ladenden Datei muss der Dateiname auch mit `" "` eingeschlossen werden, wenn der Pfad Leerzeichen enthält. Beispiel: `/F:"X:\Abc\Meine Datei.crv"`.

Der letzte Teil dieser Methode ruft die Funktion Login auf und öffnet – sofern das Login erfolgreich war – das Hauptfenster `frmTrend`.

2.3.1.2.2 Public Function GetVerString()

Parameter: *sClass As String*
sTitle As String

Rückgabe: *String*

Liefert eine Kombination aus dem übergebenen Titel und der Versionsnummer der Klasse, die über die Funktion `GetFileVersionFromClass` der Bibliothek `SSLib` ermittelt wird. Zum Beispiel ergeben die Parameter `sClass="SSLibUI.StdDlg"` und `sTitle="SSLibUI"` den Ergebnistext `"SSLibUI Version 2.10"`. Diese Funktion wird nur für den Dialog `dlgAbout` benötigt, der damit die Liste der Module ausfüllt.

2.3.1.2.3 Public Function TrimStr()

Parameter: *ByVal s As Variant*

Rückgabe: *String*

Liefert einen getrimmten String (also von Leerzeichen am Anfang und Ende befreit). Das Besondere dieser Funktion ist, dass sie gegen `NULL`[6] abgesichert ist. Enthält der übergebene Wert `NULL`, so wird ein Leerstring zurückgeliefert.

Anmerkung: Wird in `PMS_Trend` nicht verwendet.

2.3.1.2.4 Public Function TrimZero()

Parameter: *ByVal s As Variant*

Rückgabe: *String*

Löscht führende Nullen aus einem Zahlenstring. Da bei einigen Angaben in der Datenbank anstelle von numerischen Werten Zahlenstrings mit einer festen Länge verwendet werden, ist es sinnvoll, diese Strings von allen führenden Nullen zu befreien. Zur Sicherheit wird der Rest des Strings noch durch eine Konvertierung in einen `Long`-Wert und wieder zurück kontrolliert.

---

[5] 1 … 4 sind dabei die vier Ecken des ersten, und 5 … 8 des zweiten Monitors

[6] Entspricht dem `NULL`-Wert, der über die Datenbank-Schnittstelle kommen kann.

Anmerkung: Wird in **PMS_Trend** nicht verwendet.

2.3.1.2.5 Public Function FillZero()

Parameter: *ByVal s As String*
ByVal n As Integer

Rückgabe: *String*

Da bei einigen Angaben in der Datenbank anstelle von numerischen Werten Zahlenstrings mit einer festen Länge verwendet werden, ist es mit dieser Funktion einfach, einen String **s** mit führenden Nullen auf die in **n** festgelegte Länge aufzufüllen.

Ist der Originalstring bereits ausreichend land, oder sogar länger, wird er unverändert zurückgeliefert. Ebenso gilt das, wenn er den Wert **"-1"** enthält, da dies in PMS allgemein als „nicht vorhanden" gilt.

Anmerkung: Wird in **PMS_Trend** nicht verwendet.

2.3.1.2.6 Public Function PWDEncode()

Parameter: *ByVal sOriginal As String*

Rückgabe: *String*

Trivial-Verschlüsselung eines Textes. Hiermit werden die Zeichen des Strings **sOriginal** in numerische Werte umgerechnet, die dann durch einfache Rechnungen noch etwas verändert und – durch Leerzeichen getrennt – als Ergebnis zurückgegeben werden.

Diese Funktion wird dazu verwendet, Kennworte nicht völlig offen zu speichern. Die Verschlüsselung ist zwar sehr schwach, reicht aber für den Normalfall. (Siehe 2.3.1.2.7)

2.3.1.2.7 Public Function PWDDecode()

Parameter: *ByVal sCrypt As String*

Rückgabe: *String*

Trivial-Entschlüsselung eines durch **PWDEncode** (siehe 2.3.1.2.6) verschlüsselten Strings. Die Funktion erwartet einen verschlüsselten Text in **sCrypt**, entschlüsselt ihn und liefert das Ergebnis zurück. Tritt dabei ein Fehler auf, so wird ein Leerstring geliefert.

2.3.1.2.8 Public Function UserString()

Parameter: *ByVal vNewUser As Variant*
ByVal vNewDate As Variant
ByVal vUpdUser As Variant
ByVal vUpdDate As Variant
Optional ByVal sPrefix As String = ""

Rückgabe: *String*

Diese Funktion liefert aus den vier angegebenen User-, bzw. Datumswerten eine Kombination, die hauptsächlich zum Anzeigen von Erstellungs- und Änderungsdaten gedacht ist. (Siehe 2.3.1.2.9)

Anmerkung: Wird in **PMS_Trend** nicht verwendet.

2.3.1.2.9 Public Function UserAndDate()

Parameter: *ByVal sUser As Variant*
ByVal dDate As Variant
Optional ByVal sPrefix As String = ""

Rückgabe: *String*

Diese Funktion kombiniert die übergebenen Werte **sUser** und **dDate** so, dass sich ein String wie **"Benutzer am 12.03.2006"** ergibt. Außerdem kann noch ein dritter Parameter **sPrefix** übergeben werden, der dann dem anderen Text vorangestellt wird.

Hierbei wird auch auf leere Strings und ungültige Datumswerte entsprechend reagiert.

Anmerkung: Wird in **PMS_Trend** nicht verwendet.

2.3.1.2.10 Public Function IsValidPlant()

Parameter: ByVal s As String

Rückgabe: Boolean

Liefert **True**, wenn **s** ein gültiges Betriebskürzel enthält. Dazu wird in der PMS-DB einfach nachgesehen, ob der Betrieb **s** in der Tabelle **[Anlagen]** vorkommt. Wenn ja, wird **True** geliefert, ansonsten **False**.

2.3.1.2.11 Public Function Login()

Parameter: Optional ByVal bRelogin As Boolean = False

Rückgabe: Boolean

Diese Funktion verwaltet die Anmeldung des Benutzers am PMS-System. Dazu wird als erstes mit der Methode **Login** (siehe 2.4.1.2.3) des globalen Objekts **g_Security** der Anmeldedialog angezeigt und das Ergebnis abgefragt. Dabei werden sowohl Benutzername und Kennwort aus dem Security-Objekt, als auch der Parameter **bRelogin** übergeben.

Ist dieser Schalter **True**, so können im Dialog auch bei bereits bestehendem Benutzernamen und Kennwort neue Werte eingegeben werden. Anderenfalls schließt sich der Dialog sofort wieder, wenn die Angaben mit den aus der PMS-Datenbank übereinstimmen.

Wurde der Dialog abgebrochen, so wird die Funktion sofort mit **False** beendet. Anderenfalls bedeutet das, dass die Eingaben in Ordnung sind und ein Benutzer mit diesem Namen existiert. Unter anderem wird im Security-Objekt dabei auch der Default-Betrieb gesetzt. Ist dies kein gültiger Betrieb, so wird hier noch versucht, über eine andere Verknüpfung in die PMS-DB den Betrieb zu ermitteln. Schlägt dies auch fehl, so hat der Benutzer keinen und der Betrieb wird mit einem Leerstring gefüllt.

Als nächstes wird der Anmeldeserver ermittelt. Hat der Benutzer keinen Betrieb, so wird mit **LoadTrendServer** (siehe 2.3.1.2.14) versucht, einen solchen aus der Tabelle **Int_Parameter** in **PMS** zu ermitteln. Schlägt dies auch fehl, wird der Benutzer auf den Server **\\OFFSBMS100** festgelegt[7].

Konnte dem Benutzer ein Betrieb zugeordnet werden, so wird als nächstes mit **GetServerName** (siehe 2.3.1.2.19) der Anmeldeserver, sowie der Benutzername und das Kennwort, mit dem sich das Programm diesem gegenüber authentifiziert, ermittelt.

In beiden Fällen müssen die globalen Variablen **g_sDataServer**, **g_sDataUsr** und **g_sDataPwd** nun einen voll qualifizierten Servernamen, einen Benutzer und das Kennwort dazu enthalten. Wenn nicht, wird die Funktion mit **False** beendet.

Wenn ja, wird die Funktion **Authenticate** (siehe 2.3.1.2.12) aufgerufen und ihr Rückgabewert als Ergebnis von **Login** verwendet.

[7] Dieser Server liegt – im Gegensatz zu den Betriebsservern – mit einer Netzwerkkarte im DTINET-Netzwerk und mit einer zweiten in EMR-OFF.LOCAL (multi-homed). Daher kann er auch von „normalen" Arbeitsstationen erreicht werden und trotzdem die Daten der Betriebsserver darstellen.

2.3.1.2.12 Public Function Authenticate()

Parameter: —

Rückgabe: *Boolean*

Die Authentifizierung am Datenserver durchführen. Diese Funktion wird benötigt, da der Name, mit dem sich der Benutzer an der PMS-Datenbank anmeldet, nicht unbedingt auch ein gültiger Windows-Benutzername ist. Außerdem wird zum Auslesen des Verzeichnisbaums eine Netzwerkverbindung mit einem Laufwerk benötigt. Diese beiden Umstände werden darüber gelöst, dass in der Funktion Login aus den allgemeinen Daten u. a. auch ein Windowsbenutzer ermittelt wird, mit dem das Programm eine Netzwerkverbindung zum Datenserver herstellt.

Zuerst wird dazu eine eventuell bereits bestehende Verbindung von **g_sShareDrive** mit einem Laufwerk getrennt (über **UnAuthenticate**, siehe 2.3.1.2.13). Danach werden die Parameter derart bereinigt, dass daraus ein Kommando der Form

```
%WINDIR%\system32\net.exe use <Laufwerk> "\\<Server>\iTrends" "<Kennwort>" /user:<Benutzername>
```

erstellt werden kann. Dieses Kommando wird dann der Funktion **RunAndWait** (siehe 2.3.1.2.24) übergeben, die es synchron ausführt und anhand des Rückgabewertes **0** (erfolgreich) oder **<>0** (Fehler) zurückliefert. Aus diesem Ergebnis wird dann der Rückgabewert von **Authenticate** ermittelt.

2.3.1.2.13 Public Function UnAuthenticate()

Parameter: *ByVal sDrv As String*

Rückgabe: *Boolean*

Die zuvor vorgenommene Authentifizierung und das Netzlaufwerk entfernen. Dazu wird einfach der Funktion **RunAndWait** (siehe 2.3.1.2.24) das Kommando

```
%WINDIR%\system32\net.exe use <Laufwerk> /d
```

übergeben, wodurch die Verbindung des Laufwerks in **sDrv** vom Netzwerk getrennt wird. Auch hier wird das Ergebnis wieder in einen Rückgabewert umgewandelt.

2.3.1.2.14 Private Function LoadTrendServer()

Parameter: —

Rückgabe: *Boolean*

Diese Funktion liest – sofern vorhanden – aus der PMS-Tabelle **Int_Parameter** die Werte der drei Parameter **Trendserver**, **Trenduser** und **Trendpassword**[8] und speichert sie in **g_sDataServer**, **g_sDataUsr** bzw. **g_sDataPwd**.

Sie liefert **True**, wenn alle drei Parameter gefunden wurden, ansonsten **False**.

2.3.1.2.15 Public Sub ShowAbout()

Parameter: —

Rückgabe: —

Zeigt einfach nur den Info-Dialog (siehe 2.5.1.3.2) des Programms als modales Fenster an.

2.3.1.2.16 Public Function CompName()

Parameter: *ByVal sName As String*

Rückgabe: *String*

Diese Methode erstellt aus einem Computernamen, bzw. einer IP-Adresse einen normierten Namen. Dazu wird von **sName** (außer bei einer numerischen IP-Adresse) alles ab dem ersten Punkt entfernt und der Rest in Großbuchstaben zurückgeliefert.

[8] verschlüsselt

2.3.1.2.17 Public Function OnlyName()

Parameter: *ByVal s As String*

Rückgabe: *String*

Erstellt aus einem Pfadnamen in **s** den reinen Dateinamen, indem zuerst alles bis inklusive dem letzten Backslash entfernt wird. Danach wird noch alles ab dem letzten Punkt entfernt und der Rest zurückgegeben.

2.3.1.2.18 Public Function OnlyExt()

Parameter: *ByVal s As String*

Rückgabe: *String*

Extrahiert die Erweiterung aus einem Dateinamen in **s**. Dazu wird der Text ab (und inklusive) dem letzten Punkt zurückgegeben. Enthält der Dateiname keinen Punkt, so wird ein Leerstring geliefert.

2.3.1.2.19 Public Function GetServerName()

Parameter: *ByVal sBetrieb As String*

Rückgabe: *String*

Diese Funktion ermittelt den zur Zeit aktiven Datenserver eines Betriebes, der in **sBetrieb** angegeben wird. Dazu wird zuerst mit **GetPLSConnectionString** der Verbindungsstring für das ActiveX-Control **PMS_Mod** (siehe dazu eigene Dokumentation) aus der PMS-Datenbank gelesen. Wird dort kein Verbindungsstring gefunden, prüft die Funktion noch, ob einer der beiden Servernamen im Security-Objekt ausgefüllt ist und verwendet diesen dann.

Ist an dieser Stelle noch nichts vorhanden, so bricht die Funktion mit einer Meldung ab und gibt einen Leerstring zurück.

Anderenfalls wird ein Objekt vom Typ **PMS_Mod** erstellt und versucht, eine Verbindung mit dem Datenserver herzustellen. Schlägt das fehl, wird ebenfalls abgebrochen.

Bei Erfolg hat **PMS_Mod** den Server ermittelt, der momentan tatsächlich Daten liefert. Dieser wird nun aus der Eigenschaft **.HostName** übernommen und zurückgeliefert.

2.3.1.2.20 Public Function GetPLSConnectionString()

Parameter: *ByVal sBetrieb As String*

Rückgabe: *String*

Holt den Verbindungsstring zur Messdatenerfassung für den Betrieb **sBetrieb** aus der PMS-Datenbank. Dazu wird zuerst in der DB nachgesehen, ob es für den Betrieb einen Verbindungsstring gibt. Wenn ja, wird dieser zurückgegeben.

Schlägt dies dagegen fehl, so wird mit **GuessConnString** (siehe 2.3.1.2.21) ein Standard-Verbindungsstring erstellt und dieser zurückgeliefert.

2.3.1.2.21 Public Function GuessConnString()

Parameter: —

Rückgabe: *String*

Erstellt aus **UserBetrieb**, **Server1** und **Server2** des Security-Objekts einen Verbindungsstring für die Datenerfassung und liefert ihn zurück. Dieser String hat die Form

```
TYP=W;BTR=<Betrieb>;HST=<Server1>,<Server2>;USR=wwUser;PWD=wwuser
```
[9]

Fehlt eine der drei Angaben im Security-Objekt, so wird ein Leerstring zurückgegeben.

[9] zur genaueren Beschreibung der einzelnen Abschnitte siehe die Dokumentation zu **PMS_Mod**.

2.3.1.2.22 Public Function ValidStr()

Parameter: *ByVal v As Variant*
Optional ByVal sDef As String = ""

Rückgabe: *String*

Erstellt aus dem Parameter **s** einen gültigen String ohne Leerzeichen zu Beginn und am Ende. Wenn **s** gleich **NULL**[10] ist, so wird der zweite Parameter **sDef** zurückgeliefert.

2.3.1.2.23 Public Function ValidLng()

Parameter: *ByVal v As Variant*
Optional ByVal lDef As Long = 0

Rückgabe: *Long*

Funktioniert genau wie **ValidStr** (siehe 2.3.1.2.22), nur dass der Parameter v in einen Long-Wert anstelle eines Strings konvertiert wird.

2.3.1.2.24 Private Function RunAndWait()

Parameter: *ByVal sProg As String*
ByVal sParam As String

Rückgabe: *Long*

Diese Funktion ruft ein externes Programm (versteckt) auf und wartet auf seine Beendigung (synchron). Da die „offiziellen" VB-Funktionen zum Aufruf von Programmen nicht dazu in der Lage sind, ein Programm zu verstecken, außerdem nicht auf das Ende desselben warten können (nur asynchron) und daher auch nicht den Rückgabewert des Programms ermitteln können, war es notwendig, auf Standardfunktionen von Windows zurückzugreifen.

Mit **CreateProcess** wird der Prozess gestartet, mit **WaitForSingleObject** auf sein Ende gewartet und danach mit **GetExitCodeProcess** der Rückgabewert ermittelt und alle Handles dann mit **CloseHandle** geschlossen.

Zuletzt wird der Rückgabewert als Ergebnis der Funktion zurückgegeben.

2.3.1.2.25 Private Function GetFreeDrive()

Parameter: —

Rückgabe: *String*

Ermittelt das letzte unbenutzte Laufwerk auf dem aktuellen Computer. Dazu wird in einer Schleife von 90 (=**Z:**) bis 65 (=**A:**) die Windowsmethode **GetDriveType** für jeden Buchstaben aufgerufen, bis das Ergebnis entweder **DRIVE_NO_ROOT_DIR** oder **DRIVE_UNKNOWN** ist oder die Schleife beendet ist. Wurde ein ungültiges Laufwerk gefunden, so wird der entsprechende String (z. B. **Y:**) zurückgegeben. Ansonsten wird ein Leerstring geliefert.

2.3.1.2.26 Public Function GetRoot()

Parameter: —

Rückgabe: *String*

Ermittelt die Bezeichnung der Freigabe, in der die **iTrends**-Daten abgelegt sind. Dazu wird, falls die globale Variable **g_sDataServer** noch leer ist, mit **GetServerName** (siehe 2.3.1.2.19) der Datenserver ermittelt.

Anschließend wird der Servername normiert und der Freigabename **iTrends** angehängt. Ist damit noch keine gültige Freigabe zustande gekommen, so holt die Funktion aus der lokalen Registry den Eintrag "**Root Path**". Ist dies auch nicht gültig, so wird ein Leerstring zurückgegeben.

---

[10] gemeint ist hier wieder der Nullwert aus der Datenbank

2.3.2 basError

Dieses Modul stellt globale Methoden zur Fehlerbehandlung bereit.

2.3.2.1 Datenelemente

Name	Typ	Zweck
g_csErrMailRecipient	Const String	Parametername für [Int_Parameter] (Mail-Empfänger)
g_csErrMailSMTPServer	Const String	Parametername für [Int_Parameter] (SMTP-Server)
m_nMaxErrCount	Const Integer	Maximale Anzahl gleicher Meldungen (3)
m_nErrCount	Integer	Anzahl der Fehler mit gleichem Fehlercode
m_nErrCode	Long	Der zuletzt aufgetretene Fehlercode

Tabelle 8: basError - Datenelemente

2.3.2.2 Funktionen und Methoden

2.3.2.2.1 Public Sub ErrorHandler()

Parameter: *ByVal sInfo As String*
Optional ByVal vErr As Variant = Nothing

Rückgabe: —

Globale Fehlerbehandlung. Zuerst wird geprüft, ob in `vErr` ein Wert übergeben wurde. Wenn nicht, wird das globale Objekt `Err` nach `vErr` kopiert. Ist der enthaltene Fehlercode `0` (kein Fehler), wird die Methode sofort beendet.

Andernfalls wird getestet, ob dieser Fehler wiederholt aufgetreten ist (ein Beispiel hierfür wäre, dass die Datenbank nicht mehr erreicht werden kann). In diesem Fall wird der Schalter `bQuit` gesetzt, der später bewirkt, dass das gesamte Programm beendet wird.

Danach wird aus den verschiedenen Fehlerinformationen und dem Parameter `sInfo` ein String formatiert und mit `LogError` (siehe 2.3.2.2.2) in der PMS-Datenbank gespeichert.

Als nächstes werden noch alle aktuellen Datenbankfehler aufgesammelt und ebenfalls mit `LogError` gespeichert.

Nun wird, falls `bQuit` gesetzt ist, das Programm beendet.

Ist `bQuit` jedoch gelöscht, wird ein Dialog vom Typ `dlgErrHndl` (siehe 2.5.3) erstellt, mit den bisher gesammelten Informationen gefüllt und angezeigt.

2.3.2.2.2 Public Sub LogError()

Parameter: *ByVal lErrNr As Long*
ByVal sErrSource As String
ByVal sErrLocation As String
ByVal sErrDescription As String

Rückgabe: —

Fehler in die PMS-Tabelle `Int_ErrorLog` schreiben. Diese Funktion speichert Fehlermeldungen mit allen übergebenen Parametern in der PMS-Datenbank, wo sie ggf. später analysiert werden können. Ist die Datenbank nicht erreichbar, so wird anstelle des Eintrags eine normale `MsgBox` angezeigt.

2.3.3 basSQL

In diesem Modul sind alle SQL-Kommandos, teilweise als Formatierungsvorlagen (siehe 2.3.3.2.1), enthalten; weiterhin Methoden, die sich direkt auf die Datenbank beziehen.

Es ist zur Übersichtlichkeit von Relevanz, die SQL-Kommandos nicht im gesamten Quellcode zu verstreuen, sondern sie in einem Modul zu konzentrieren.

2.3.3.1 Datenelemente

Name	Typ	Zweck
sSqlConnect	Const String	Ein Template zur Erstellung eines Connectionstrings für die DB. Muss mit Servername, Benutzer und Kennwort gefüllt werden.
sSqlP_CheckConnection	Const String	Ein einfaches und schnelles Kommando zum Prüfen, ob die DB erreichbar ist. Wird in **CheckPMS** verwendet.
sSqlGetUserDescr1	Const String	Holt den Benutzernamen und die Beschreibung.
sSqlGetUserDescr2	Const String	Wie **sSqlGetUserDescr1**, jedoch ohne Kennwort.
sSqlGetUserBetrieb	Const String	Liest den Betrieb eines Benutzers
sSqlGetUser	Const String	Holt die Benutzerinformationen, die in CSecurity gespeichert werden.
sSqlGetSecurityItem	Const String	Holt alle Items, auf die der Benutzer Zugriff hat.
sSqlGetSecurityUnits	Const String	Liest die Betriebe für einen Benutzer.
sSqlGetAnlConnect	Const String	Prüft, ob der angegebene Betrieb in der DB bekannt ist.
sSqlGetTrendServer	Const String	Holt die Angaben zum Datenserver aus der DB.
sSqlInsertError	Const String	Speichert die Angaben zu einem Fehler in der DB.
sSqlGetParameter	Const String	Liefert den Wert eines internen Parmeters aus der Tabelle **Int_Parameter**.
sSqlGetServer	Const String	wird nicht benutzt.
sSqlGetConnect	Const String	Holt den Connection-String zur Messdatenerfassung aus der DB.
sSqlGetPlantServers	Const String	Holt alle bekannten Server für einen Betrieb.
m_Conn	Connection	Das interne Connection-Object zur PMS-DB.
m_sDBServer	String	Eine Kopie des Namens des Datenbankservers

Tabelle 9: basSQL - Datenelemente

2.3.3.2 Funktionen und Methoden

2.3.3.2.1 Public Function FormatSQL()

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

Rückgabe: *String*

Formatiert einen String anhand eines Formatstrings und einer Liste von Parametern und gibt das Ergebnis als String zurück.

Dabei wird als 1. Parameter ein Formatstring übergeben, der nummerierte Felder in der Form **%1** bis **%n** enthalten kann. Diese Felder werden mit den übrigen (optionalen) Parametern aus dem Feld **par()**[11] aufgefüllt. Enthält der Formatstring die gleiche **%**-Nummer mehrmals, so wird auch der entsprechende Parameter mehrmals eingefügt.

Achtung: Werden mehr Parameter angegeben, als Felder vorhanden sind, so werden die überzähligen ignoriert. Sind dagegen zuwenig Parameter angegeben, bleiben die unbenutzten Platzhalter unverändert!

Wird das Zeichen **%** als Literal benötigt (bei Abfragen mit **LIKE**), so kann ersatzweise ein **#** anstelle des **%** im Formatstring eingesetzt werden.

[11] Diese Parameter werden als VARIANT, also als beliebiger Typ, erwartet und lassen sich einfach in beliebiger(?) Menge einfügen, z. B. liefert **FormatSQL("SELECT %1, %2 FROM %4%2", 123, "abc", "TabX")** den String "**SELECT 123, abc FROM TabXabc**" zurück.

2.3.3.2.2 Public Function CheckPMS()

Parameter: —

Rückgabe: *Boolean*

Prüft, ob bereits eine Verbindung zur PMS-Datenbank vorhanden ist. Wenn ja, wird mit einem einfachen **SELECT** geprüft, ob die Datenbank reagiert. Schlägt dies fehl, versucht die Funktion mit **ReconnectPMS** (siehe 2.3.3.2.3) die Verbindung herzustellen und liefert deren Ergebnis zurück. Funktioniert der Test, so ist die Verbindung in Ordnung und **CheckPMS** liefert **True** zurück.

2.3.3.2.3 Public Function ReconnectPMS()

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

Rückgabe: *Boolean*

Schließt die aktuelle Verbindung zur PMS-Datenbank und erstellt sie neu. Funktioniert dieses, ist die Verbindung (in der lokalen Variable **m_Conn**) bereits geöffnet und die **ReconnectPMS** liefert **True**. Ansonsten ist **m_Conn** gelöscht (= **Nothing**) und die Funktion liefert **False**.

Wenn ein String als Parameter übergeben wird, so wird dieser als Verbindungsstring verwendet. Ansonsten (Normalfall) werden die Verbindungsinformationen aus der Registry gelesen und damit ein Verbindungsstring aufgebaut.

Außerdem wird, falls das Hauptfenster bereits besteht, der Datenbankserver in der Statusleiste angezeigt.

2.3.3.2.4 Public Sub ClearConn()

Parameter: —

Rückgabe: —

Wenn das lokale Objekt **m_Conn** existiert, wird es ggf. geschlossen und vernichtet. Nach dem Aufruf dieser Methode ist die Verbindung zur PMS-Datenbank auf jeden Fall getrennt und muss vor dem nächsten Zugriff z. B. mit **GetConn** (siehe 2.3.3.2.5) wieder erstellt werden.

2.3.3.2.5 Public Function GetConn()

Parameter: —

Rückgabe: *ADODB.Connection*

Liefert das interne Objekt **m_Conn** zurück, sofern es existiert, bzw. erstellt werden kann. Im Fehlerfall liefert die Funktion **Nothing** zurück.

Dies ist der ‚offizielle' Zugang zur Datenbankverbindung. Durch diese Art der Kapselung wird sichergestellt, dass immer eine gültige Verbindung verwendet wird.

2.3.3.2.6 Public Function GetDbInfos()

Parameter: *ByRef sServer As String*
ByRef sUser As String
ByRef sPwd As String
ByRef sCatalog As String

Rückgabe: *Boolean*

Liest die vier Datenbankangaben aus der Registry. Hierbei wird zuerst versucht, eine ‚lokale Kopie' der Werte im Key **HKLM\Software\KoSa\PMS\Trend** zu lesen.

Schlägt dies fehl, werden die Parameter aus **HKLM\Software\KoSa\PMS\SQLDB** gelesen, wo die Datenbank-Angaben für alle PMS-Programme gespeichert werden. Anschließend werden diese mit **SaveDbInfos** im o. g. Key nochmals gespeichert. Bei beiden Zugriffen wird das Kennwort verschlüsselt erwartet (siehe dazu 2.3.1.2.7).

2.3.3.2.7 Public Sub SaveDbInfos()

Parameter: *ByVal sServer As String*
ByVal sUser As String
ByVal sPwd As String
ByVal sCatalog As String

Rückgabe: —

Speichert die übergebenen Parameter als Datenbankinformationen im Registry-Key **HKLM\Software\KoSa\PMS\SQLDB** ab. Dabei wird das Kennwort verschlüsselt (siehe dazu 2.3.1.2.6).

2.3.3.2.8 Public Function GetDbServer()

Parameter: —

Rückgabe: *String*

Liefert den Namen des Datenbankservers (Variable **m_sDBServer**) zurück.

2.3.3.2.9 Public Function GetParameter()

Parameter: *ByVal sParameter As String*

Rückgabe: *Variant*

Gibt den Wert eines internen Parameters mit dem Namen in **sParameter** aus der Tabelle **Int_Parameter** in der Datenbank **PMS** zurück. Wenn der Parameter dort nicht existiert, wird **Null** zurückgegeben.

2.3.4 MonitorInfo

Dieses Modul ermittelt die Anzahl der vorhandenen Monitore, sowie die Abmessungen derselben. Da speziell **PMS_Display** dafür vorgesehen ist, pro Monitor vier einzelne Kurvenfenster darzustellen, ist die Ermittlung dieser Parameter unabdingbar.

Um die Koordinaten für ein Fenster zu ermitteln, muss nur die Funktion **Coordinates** aufgerufen werden. Die komplette Initialisierung geschieht intern.

2.3.4.1 Datenelemente

Name	Typ	Zweck
SM_CMONITORS	Const Long	Parameter für die Windows-Funktion GetSystemMetrics()
RECT	Type	Struktur, die Long-Werte für alle vier Seiten eines Rechtecks enthält
MONITORINFO	Type	Struktur, die für die Funktion GetSystemMetrics() benötigt wird.
m_nNum	Integer	Enthält die Anzahl der gefundenen Monitore.
m_Rect()	Array RECT	Enthält für jeden vorhandenen Monitor je einen Satz Abmessungen.
CopyMemory EnumDisplayMonitors GetSystemMetrics GetMonitorInfo	Function	Windows-Funktionen zur Ermittlung der vorhandenen Monitore und ihrer Abmessungen, sowie Hilfsfunktionen dazu.

Tabelle 10: MonitorInfo - Datenelemente

2.3.4.2 Funktionen und Methoden

2.3.4.2.1 Public Function SampleMonitorInfo()

Parameter: —

Rückgabe: *Long*

Sammelt Monitor-Größen auf, liefert die Anzahl an Monitoren zurück. Zuerst wird das Array `m_Rect` auf die Anzahl der vorhandenen Monitor dimensioniert (0-basiert). Danach wird die Windows-Funktion `EnumDisplayMonitors` aufgerufen, die als 3. Parameter die Adresse der Funktion `MonitorEnumProc` (siehe 2.3.4.2.3) mitgeteilt bekommt, die dann im Laufe der Analyse für jeden vorhandenen Monitor einmal aufgerufen wird. Als Rückgabewert wird die Anzahl der Monitore (wird von `MonitorEnumProc` in `m_nNum` gespeichert) verwendet.

2.3.4.2.2 Public Function Coordinates()

Parameter: *ByVal idx As Integer*

Rückgabe: *RECT*

Liefert die Koordinaten für jeweils ein Viertel eines Monitors, das durch den `idx` definiert ist. Dabei sind die Fenster pro Monitor folgendermaßen angeordnet:

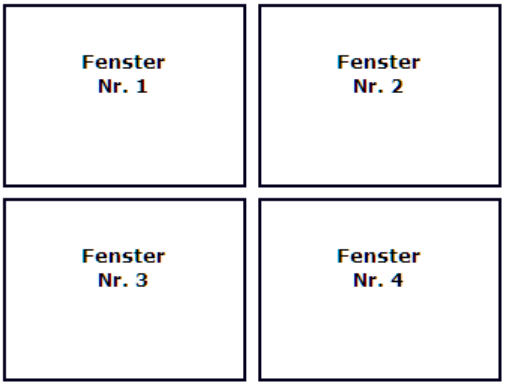


Abbildung 1: Fensternummerierung

Um die Koordinaten für Fenster auf dem zweiten Monitor zu erhalten, wird zur Nummer einfach 4 (`idx` ist also 5 … 8) addiert.

In der Funktion werden als erstes der Index um `1` vermindert, um auf die `0`-basierte Indizierung zu kommen. Dann werden jedes Mal mit `SampleMonitorInfo` die aktuellen Informationen zu den Monitoren ermittelt, um auch bei dynamischen Änderungen aktuell zu bleiben. Danach wird kontrolliert, ob überhaupt ein Monitor gefunden wurde und `idx` größer oder gleich `0` ist. Wenn nicht, wird die Funktion sofort beendet.

Anschließend wird von `idx` solange `4` subtrahiert, bis der Wert im gültigen Bereich ist[12]. Dann wird der Rückgabewert aus dem Array `m_Rect` kopiert, das dem gewünschten Monitor entspricht und je nach Lage die entsprechenden Koordinaten berichtigt.

[12] Also bei einem Monitor 0…3, bei zwei Monitoren 0…7, etc.

2.3.4.2.3 Public Function MonitorEnumProc()

Parameter: *ByVal hMonitor As Long*
ByVal hdcMonitor As Long
ByVal rcMonitor As Long
ByVal dwData As Long

Rückgabe: *Long*

Diese Funktion ist der Callback für die Windows-Funktion `EnumDisplayMonitors`. Sie wird für jeden vorhandenen Monitor einmal aufgerufen. Als erstes wird mit `CopyMemory` der Parameter `rcMonitor` in den aktuellen Index von `m_Rect` kopiert.

Danach wird versucht, mit der Funktion `GetMonitorInfo` den Arbeitsbereich des Monitors zu ermitteln[13]. Da diese Funktion (aus unbekannten Gründen) nicht immer einen sinnvollen Wert liefert, muss ihr Rückgabewert überprüft werden. Nur wenn dieser ungleich `0` ist, wird das Rechteck aus `mi` in den aktuellen Index von `m_Rect` kopiert.

Abschließend wird der aktuelle Index `m_nNum` um eins erhöht und `1` zurückgegeben, damit `EnumDisplayMonitors` weitermacht[14].

2.3.5 MultiHook

Dieses Modul stellt die normalerweise in VB 6 nicht vorhandene Möglichkeit zur Verfügung, sich in Windows-Messages einzuhängen und somit auch von den VB-Entwicklern „übersehene" Funktionalitäten zu realisieren.

In PMS_Trend und PMS_Display wird dies zum Aktivieren der Standardfunktionen „minimale und maximale Fenstergröße" verwendet, da alle anderen Ansätze nur ein sehr trauriges Abbild davon schaffen.

ACHTUNG: *Da diese Vorgänge unter Umgehung des ‚VB-Sandkastens' direkt am System eingreifen können, ist die Verwendung nur unter größter Vorsicht zu empfehlen! Der Entwickler sollte sich mit solchen Methoden auskennen und trotzdem auf Programmabstürze gefasst sein.*

2.3.5.1 Datenelemente

Name	Typ	Zweck
POINTAPI	Type	Unterelement von `MINMAXINFO`
MINMAXINFO	Type	Windows-Struktur; wird in `GetMinMax` verwendet
GWL_WNDPROC WM_GETMINMAXINFO	Const Long	Konstanten für die Windows-Funktionen
m_HookedForms	Collection	Liste der in MultiHook verwalteten Fenster
m_OldWndProcs	Collection	Liste der Pointer auf die Original-WndProcs
GetModuleFileName CopyMemory CallWindowProc SetWindowLong	Function	Windows-Funktionen zum Einklinken in Standard-Messages, sowie Hilfsfunktionen dazu.

Tabelle 11: MultiHook - Datenelemente

2.3.5.2 Funktionen und Methoden

2.3.5.2.1 Public Function IsInIDE()

Parameter: —

Rückgabe: *Boolean*

Diese Funktion prüft, ob das Programm zur Zeit innerhalb der IDE gestartet wurde (Rückgabe `True`), oder ob es als eigenständiges Programm läuft (Rückgabe `False`).

---

[13] Der Arbeitsbereich ist die Monitorfläche abzüglich der Systembereiche, die durch Taskleisten, u. ä. belegt sind.
[14] Würde stattdessen `0` zurückgegeben, würde `EnumDisplayMonitors` abbrechen.

Dazu wird mit **GetModuleFileName** der Dateiname des aktuell laufenden Moduls ermittelt und geprüft, ob der Text „VB6.EXE" enthalten ist. Man sollte sein Programm also nicht unbedingt „**MeinProgrammUnterVB6.exe**" oder so ähnlich nennen.

Dieser Test ist notwendig, da die Auswirkungen von veränderten Message-Hooks unter der IDE mindestens dazu führen, dass die VB-IDE abstürzt. Das ist besonders verheerend, wenn die letzten Änderungen noch nicht gespeichert wurden…

2.3.5.2.2 Public Sub HookForm()

Parameter: *frm As Form*

Rückgabe: *—*

Mit dieser Methode wird das angegebene Fenster in die Liste der zu verwaltenden Fenster aufgenommen[15]. Als erstes wird hier getestet, ob das Programm unter der IDE läuft. Wenn ja, wird die Routine sofort beendet, um Systemabstürze zu vermeiden.

Danach wird das Fenster in die Liste verwalteter Fenster in **m_HookedForms** aufgenommen und danach mit der Methode **SetWindowLong** die aktuelle Window-Prozedur auf **newWndProc** (siehe 2.3.5.2.5) gestellt. Der Rückgabewert der Methode ist der bisherige Zeiger; er wird in der Liste **m_OldWndProcs** gespeichert, damit das System später wieder normalisiert werden kann.

2.3.5.2.3 Public Sub UnhookForm()

Parameter: *frm As Form*

Rückgabe: *—*

Diese Methode nimmt die Spezialbehandlung eines Fensters wieder zurück. Auch hier wird unter der IDE sofort abgebrochen.

Danach wird nachgesehen, ob das Fenster überhaupt in der Liste in **m_OldWndProcs** ist. Wenn nicht, wird auch einfach abgebrochen. Ansonsten wird der Originalzeiger wieder hergestellt und das Fenster aus beiden Listen gelöscht.

2.3.5.2.4 Public Sub UnhookAllForms()

Parameter: *—*

Rückgabe: *—*

Hiermit werden alle eventuell bestehenden Hooks entfernt. Natürlich ist auch hier wieder der Test mit **IsInIDE** obligatorisch.

Danach werden alle Fenster, die in der Liste **m_HookedForms** enthalten sind, mit **UnhookForm** desaktiviert.

2.3.5.2.5 Private Function newWndProc()

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

Rückgabe: *Long*

Diese Funktion ist der Callback, in dem die Windows-Messages abgefangen werden. Diese Funktion wird nicht vom normalen Programm, sondern vom System aufgerufen.

Als erstes wird geprüft, ob das Fenster, dessen Window-Prozedur behandelt werden soll, in **m_OldWndProcs** enthalten ist. Wenn ja, wird versucht, **Form_Message** (siehe 2.3.5.2.8) im aktuellen Fenster aufzurufen. Existiert diese Funktion und liefert **True** zurück, wird dem System durch die Rückgabe von 0 mitgeteilt, dass die aktuelle Message abgearbeitet ist.

---

[15] Der Aufruf sollte vorzugsweise direkt nach dem Laden eines Fensters, also zu Beginn der Methode **Form_Load**, in der Form „**HookForm Me**" vorgenommen werden.

Wenn **Form_Message** nicht existiert, oder **False** zurückliefert, wird mittels **CallWindowProc** die Originalmethode aus der Liste **m_OldWndProcs** aufgerufen und deren Rückgabewert ans System übermittelt.

2.3.5.2.6 Public Sub GetMinMax()

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

Rückgabe: —

Diese Methode dient nur dazu, die Min- und optional Max-Größe des Fensters festzulegen. Sie kann aus der Fenster-Prozedur **Form_Message** (siehe 2.3.5.2.8) aufgerufen werden.

Dazu wird eine Struktur des Typs **MINMAXINFO** erstellt, in der die minimale (in **MinX**/**MinY**) und optional auch die Maximalgröße (in **MaxX**/**MaxY**) des Fensters eingetragen wird. Danach wird diese Struktur mit **CopyMemory** auf die Adresse kopiert, auf die **MinMax** zeigt.

*ACHTUNG: Der Parameter **MinMax** wird hier als Systemzeiger verwendet. Daher kann es zu „interessanten[16]" Effekten führen, wenn diese Methode nicht mit den richtigen Parametern aufgerufen wird!*

2.3.5.2.7 Public Sub GetMinMaxTwips()

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

Rückgabe: —

Diese Methode bewirkt das gleiche wie **GetMinMax**, jedoch mit Twips (der seltsamen VB-internen Maßeinheit) anstatt mit Pixel. Dabei werden die Parameter **MinX**, **MinY**, **MaxX** und **MaxY** vor dem Weiterverarbeiten zuerst in Pixel umgerechnet.

2.3.5.2.8 Public Function Form_Message()

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

Rückgabe: *Boolean*

Diese Funktion ist nicht im Modul **MultiHook** vorhanden, sondern muss von dem aufrufenden Fenster bereitgestellt werden. Sie wird im Laufe von **newWndProc** aufgerufen, um die Bearbeitung der Windows-Messages zu ermöglichen. Die Funktion sollte **True** zurückgeben, wenn sie die in **uMsg** übermittelte Nachricht bearbeitet hat, und **False**, wenn das System diese Nachricht übernehmen soll.

Ein Beispiel (übernommen aus dem Hauptfenster **frmTrend**):

```
Public Function Form_Message(uMsg As Long, wParam As Long, lParam As Long) As Boolean
  Form_Message = False
  If uMsg = WM_GETMINMAXINFO Then
    GetMinMax lParam, 320, 200  ' Mindestgröße bei Festpositionierung festlegen
    Form_Message = True         ' Message wurde abgearbeitet
  End If
End Function
```

Code 1: Beispiel für Form_Message

[16] Man denke hierbei an den alten chinesischen Fluch „Mögest Du in interessanten Zeiten leben!"

Hier wird zuerst der Rückgabewert auf **False** gestellt. Danach wird getestet, ob die gesendete Nachricht **WM_GETMINMAXINFO**[17] ist. Wenn ja, wird die Mindestgröße über **GetMinMax** auf 320 × 200 Pixel festgelegt und **True** als Ergebnis zurückgegeben, wodurch die Nachricht als „bearbeitet" angenommen wird.

[17] Bei dieser Nachricht fragt das System das Fenster, ob Minimal- oder Maximalgrößen festzulegen sind.

2.4 Klassen – Detail

2.4.1 CSecurity

Diese Klasse dient dazu, die Authentifizierung eines Benutzers inkl. aller Rechte zu ermitteln und zu halten. Weiterhin hält sie noch einige benutzerbezogene Datenelemente, die jederzeit abgefragt werden können. Sie wird einmal zu Beginn des Programms initialisiert.

2.4.1.1 Datenelemente

Name	Typ	Zweck
m_sDefUnit	String	enthält den Default-Betrieb für diesen Benutzer (kann leer sein; wird hier nicht verwendet)
m_sBetrieb	String	enthält den Standard-Betrieb für diesen Benutzer (kann leer sein)
m_sFullname	String	enthält den kompletten Namen des Benutzers.
m_sName	String	enthält den Anmeldenamen des Benutzers.
m_sPassword	String	enthält das Kennwort des Benutzers.
m_sServer1	String	enthält den 1. Datenserver des Standard-Betriebs für diesen Benutzer (kann leer sein)
m_sServer2	String	enthält den 2. Datenserver des Standard-Betriebs für diesen Benutzer (kann leer sein)
m_nAccess	Long	binärer Wert, der die Berechtigungen für PMS_Trend und PMS_Display enthält
m_colEnabledItems	Collection	enthält die Namen der Funktionen, die (in `PMS_Client` und `PMS_Stammdaten`) für diesen Benutzer freigegeben sind.
m_rsUnits	ADODB.Recordset	enthält eine Liste aller erlaubten Betriebe sowie Default-Betriebe für diesen Benutzer

Tabelle 12: CSecurity - Datenelemente

2.4.1.2 Funktionen und Methoden

2.4.1.2.1 Private Sub Class_Initialize()

Parameter: —

Rückgabe: —

Diese Standard-Methode zur Initialisierung der Klasse erstellt das Objekt `m_colEnabledItems` neu.

2.4.1.2.2 Private Sub Class_Terminate()

Parameter: —

Rückgabe: —

Diese Standard-Methode zum Zerstören der Klasse löscht das Objekt `m_colEnabledItems`, indem es auf `Nothing` gesetzt wird.

2.4.1.2.3 Public Function Login()

Parameter: *ByVal sUser As String*
ByVal sPwd As String
Optional ByVal bRelogin As Boolean = False

Rückgabe: *Boolean*

Login für einen Benutzer durchführen. Dazu wird der Dialog `dlgLogin` (siehe 2.5.4.3.4) angezeigt, der die Eingaben ermöglicht und verifiziert. Die drei Parameter werden dabei an die entsprechenden Properties des Dialogs weitergegeben. Sind Benutzer (`sUser`) und Kennwort (`sPwd`) gültig (z.B. durch Kommandozeilenparameter), so wird der Dialog zwar kurz angezeigt, jedoch nur dann eine Eingabe ermöglicht, wenn auch `bRelogin` `True` ist. Anderenfalls schließt sich der Dialog sofort wieder und meldet OK zurück.

Wird der Dialog dagegen abgebrochen, so wird auch die Funktion mit dem Ergebnis **False** beendet. Ansonsten wird der Dialog entsorgt und danach die entsprechenden Daten für den Benutzer aus der Datenbank geholt[18]. Der Rückgabewert wird in diesem Fall auf **True** gestellt und mit **RequeryItems** die Namen der erlaubten Funktionen aus der Datenbank gelesen.

2.4.1.2.4 Public Sub RequeryItems()

Parameter: —

Rückgabe: —

Diese Methode füllt die Collection **m_colEnabledItems** mit den Namen der Funktionen, die (in **PMS_Client** und **PMS_Stammdaten**) für diesen Benutzer freigegeben sind. Diese Liste wird in PMS_Trend und PMS_Display zwar nicht benötigt, da das **CSecurity**-Modul aber in allen PMS-Programmen identisch übernommen wird, wurde diese Funktion belassen.

Danach wird noch das Recordset **m_rsUnits** mit den erlaubten Betrieben gefüllt und der Standardbetrieb in **m_sDefUnit** abgelegt (falls in der Datenbank ein Standardbetrieb festgelegt wurde).

2.4.1.2.5 Public Function CheckItem()

Parameter: *ByVal sItem As String*

Rückgabe: *Boolean*

Prüfen, ob die übergebene Funktionalität für den aktuellen Benutzer freigegeben (in der Collection **m_colEnabledItems** vorhanden) ist. Wenn ja, liefert die Funktion **True**, sonst **False**. Siehe dazu auch 2.4.1.2.8, **SingleItemAllowed**.

2.4.1.2.6 Public Sub AddItem()

Parameter: *ByVal sItem As String*

Rückgabe: —

Fügt **m_colEnabledItems** einen neuen Eintrag hinzu (Dieser darf noch nicht vorhanden sein).

2.4.1.2.7 Public Function CheckUnit()

Parameter: *ByVal sUnit As String*

Rückgabe: *Boolean*

Prüft, ob diesem Benutzer der übergebene Text als Standardbetrieb zugeordnet ist. Wenn ja, liefert die Funktion **True**, sonst **False**.

2.4.1.2.8 Public Function SingleItemAllowed()

Parameter: *ByVal sItem As String*

Rückgabe: *Boolean*

Ähnlich wie **CheckItem** (siehe 2.4.1.2.5) prüft auch diese Methode, ob die übergebene Funktionalität für den aktuellen Benutzer freigegeben ist. Allerdings wird hier, wenn das nicht der Fall ist, der Dialog **dlgGetUser** angezeigt, in dem ein Benutzer[19] und Kennwort mit ausreichenden Rechten eingegeben werden kann. Wurde ein gültiges Benutzer/Kennwort-Paar eingegeben, so wird die Funktionalität für diesen Aufruf freigegeben. Beim nächsten Aufruf muss die Freischaltung allerdings erneut erfolgen.

---

[18] Der Dialog kann nur dann mit OK beendet werden, wenn ein gültiger Benutzername und ggf. das zugehörige Kennwort eingegeben wurde.

[19] Der Benutzername (nicht das Kennwort!) wird dabei in einer statischen Variable gehalten, so dass bei einem zweiten Aufruf nur noch das Kennwort eingegeben werden muss.

2.4.1.2.9 Public Function GetUserUnits()

Parameter: —

Rückgabe: *ADODB.Recordset*

Liefert eine Kopie des Recordsets **m_rsUnits** zurück.

2.4.1.2.10 Public Function GetUserDefUnit()

Parameter: —

Rückgabe: *String*

Liefert den dem Benutzer zugeordneten Standardbetrieb (**m_sDefUnit**) zurück.

2.4.1.3 Attribute

2.4.1.3.1 UserBetrieb

Typ: *String*

Zugriff: *Lesen & Schreiben*

Enthält das Kürzel des Betriebs, der dem Benutzer zugewiesen ist.

2.4.1.3.2 UserFullName

Typ: *String*

Zugriff: *nur Lesen*

Enthält den ausgeschriebenen Namen des Benutzers (die Langform, sofern sie in der Datenbank eingetragen ist).

2.4.1.3.3 UserName

Typ: *String*

Zugriff: *Lesen & Schreiben*

Enthält den Anmeldenamen des Benutzers.

2.4.1.3.4 UserPassword

Typ: *String*

Zugriff: *Lesen & Schreiben*

Enthält das Kennwort des Benutzers.

2.4.1.3.5 Server1

Typ: *String*

Zugriff: *nur Lesen*

Enthält den Maschinennamen des ersten Servers, der zum Betrieb des Benutzers gehört.

2.4.1.3.6 Server2

Typ: *String*

Zugriff: *nur Lesen*

Enthält den Maschinennamen des zweiten Servers, der zum Betrieb des Benutzers gehört.

2.4.1.3.7 Access

Typ: *Long*

Zugriff: *nur Lesen*

Enthält eine Anzahl von Bits, die jeweils bestimmte Funktionen in PMS_Trend freigeben. Dieser Wert wird in der Stammdatenverwaltung eingestellt.

Die Bits haben die folgende Bedeutung:

Bit	Wert	Bezeichnung	Bedeutung
0	&H01	secUserA	Benutzer gehört zur Schicht A … D. Diese dürfen Trends ausschließlich im eigenen Verzeichnis speichern (z. B. in `iTrends\P2D\p2dUser_C`)
1	&H02	secUserB	
2	&H04	secUserC	
3	&H08	secUserD	
4	&H10	secUserMSR	Benutzer gehört zu den Mess- und Regeltechnikern
5	&H20	---	unbenutzt
6	&H40	secPowerUser	Der Benutzer darf auch Trends im Basisverzeichnis speichern
7	&H80	secAdministrator	Dieser Benutzer hat vollen Zugang zu allen Funktionen

Tabelle 13: Berechtigungen

Die Bits sind in der Enum `eSecurity` (siehe 2.3.1.1) in `basMain` definiert und können auch kombiniert werden.

2.4.2 CSplitter

Diese Klasse erstellt ein Control, das dazu verwendet wird, die Trennung zwischen zwei neben- oder übereinander liegenden Controls zu verschieben[20]. Dazu werden in der Klasse mehrere Controls kombiniert, um den gewünschten Effekt zu erzielen[21]. Dazu müssen jedoch vom aufrufenden Fenster die folgenden Elemente zur Verfügung gestellt werden:

1. Eine PictureBox[22] als Container, in dem die Controls dargestellt werden.
2. Eine weitere PictureBox als Trennlinie, die mit der Maus bewegt werden kann.
3. Die beiden Controls, die links und rechts, bzw. ober- und unterhalb der Trennlinie eingepasst werden.

Bei den Elementen 2. und 3. sind Positionierung und Größe gleichgültig, da sie von der Klasse später im Container arrangiert werden. Allerdings müssen alle Elemente im Container angeordnet sein.

Damit diese Klasse funktioniert, müssen die folgenden Events der Trennlinie (`m_SplitBar`) im aufrufenden Fenster behandelt werden:

```
Private Sub m_SplitBar_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
  m_Split.MouseDown Button, x
End Sub

Private Sub m_SplitBar_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
  m_Split.MouseMove Button, x
End Sub

Private Sub m_SplitBar_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
  m_Split.MouseUp
End Sub
```

Code 2: Eventhandler für CSplitter

Ohne diese Weiterleitung der Events in die Klasse `m_Split` passiert gar nichts. Es können natürlich noch weitere Aktionen ausgeführt werden, die hier dargestellten sind jedoch unbedingt notwendig.

---

[20] So wie beispielsweise im normalen Windows-Explorer zwischen den Ordnern auf der linken und den Dateien auf der rechten Seite.

[21] Es würde sich empfehlen, hier ein eigenes „echtes" Control zu erstellen, das die Eigenschaften besser kapselt.

[22] alternativ kann auch die Form des Fensters selbst verwendet werden. Dann bleibt allerdings kein Platz mehr für weitere Elemente, da der innere Bereich des Containers komplett ausgefüllt wird.

2.4.2.1 Datenelemente

Name	Typ	Zweck
SPLT_SIZE	Const Long	Default-Breite des Splitter-Controls in Twips.
ACTIVE_COLOR	Const Long	Default-Farbe des Splitter-Controls beim Draggen.
INACTIVE_COLOR	Const Long	Default-Farbe des Splitter-Controls im Normalzustand.
CTRL_OFFSET	Const Long	Default-Abstand der Controls zum Rand des Containers.
NO_POSITION	Const Long	Pseudo-Position, wenn inaktiv.
DEF_MARGIN	Const Long	Default-Minimum für die Controls.
INVALID	Const Long	Ungültiger Wert, der bei noch nicht positionierter Trennlinie zurückgegeben wird.
m_ctPar	Control	Das Container-Control
m_ctSplit	Control	Das Trennlinien-Control
m_ctFirst	Control	Das linke, bzw. obere Control
m_ctSecond	Control	Das rechte, bzw. untere Control
m_lCurr	Long	Die aktuelle Position.
m_lMarginLT	Long	Minimum links oder oben
m_lMarginRB	Long	Minimum rechts oder unten
m_lActiveColor	Long	Die Farbe des Splitter-Controls beim Draggen.
m_lInactiveColor	Long	Die Farbe des Splitter-Controls im Normalzustand.
m_lSize	Long	
m_lFormSize	Long	
m_lOffset	Long	Der Abstand der Controls zum Rand des Containers.
m_bShowDirect	Boolean	Wenn **True**, werden die Controls sofort mitbewegt (Live), ansonsten erst beim Loslassen der Maustaste.
m_bHor	Boolean	Wenn **True**, wird die Trennlinie horizontal dargestellt, ansonsten senkrecht.

Tabelle 14: CSplitter - Datenelemente

2.4.2.2 Funktionen und Methoden

2.4.2.2.1 Private Sub Class_Initialize()

Parameter: —

Rückgabe: —

Initialisiert alle Variablen auf die Standardwerte.

2.4.2.2.2 Public Sub InitControls()

Parameter:
ByRef ctContainer As Control
ByRef ctSplit As Control
ByRef ctFirst As Control
ByRef ctSecond As Control
Optional ByVal bHorizontal As Boolean = False

Rückgabe: —

Diese Methode initialisiert den Splitter mit den angegebenen Werten. Dabei werden die Parameter mit **AllValid** (siehe 2.4.2.2.9) geprüft und bei Fehlern eine definierte Fehlermeldung erzeugt. Wichtig ist hier auch, dass das Control, dass als Trennleiste verwendet wird, in der Darstellungsreihenfolge (**ZOrder**) ganz nach vorn gebracht wird und der Mauszeiger der Trennleiste je nach Orientierung auf ↔ bzw. ↕ eingestellt wird.

2.4.2.2.3 Public Sub MouseDown()

Parameter:
ByVal Button As Integer
ByVal lPos As Long

Rückgabe: —

Maus-Behandlung: Button wurde gedrückt. Als erstes wird mit **AllValid** (siehe 2.4.2.2.9) geprüft, ob alle Controls gültig sind, und sofort zurückgekehrt, falls nicht.

Wenn die Trennlinie desaktiviert ist – der Mauszeiger ist in diesem Fall **vbNormal** – wird die Methode ebenfalls sofort beendet.

Ansonsten wird, wenn der linke Mausbutton[23] gedrückt wurde, die Farbe der Trennlinie auf „Aktiv" gestellt und die aktuelle Position (aus **lPos**) gemerkt.

Wurde dagegen der zweite Mausbutton gedrückt, so wird mit der Methode **MouseUp** (siehe 2.4.2.2.5) der gesamte Status und die Darstellung wieder auf Normal gestellt[24].

2.4.2.2.4 Public Sub MouseMove()

Parameter: *ByVal Button As Integer*
ByVal lPos As Long

Rückgabe: —

Maus-Behandlung: Maus wurde bewegt. Auch hier wird als erstes getestet, ob alle Bedingungen erfüllt sind. Falls nicht, wird die Methode sofort beendet. Der nächste Test prüft, ob der linke Mausbutton gedrückt ist und die Trennlinie aktiviert ist.

Danach wird die Position mit **ValidatePos** validiert. Entspricht die so ermittelte Position der bisherigen, wird die Methode beendet, sonst die neue Position in **m_lCurr** gespeichert. Wenn **m_bShowDirect** gesetzt ist, werden mit **IntReposCtrls** (siehe 2.4.2.2.7) alle Controls neu positioniert, ansonsten nur die Trennlinie an die neue Position gesetzt.

2.4.2.2.5 Public Sub MouseUp()

Parameter: —

Rückgabe: —

Maus-Behandlung: Button wurde losgelassen. Auch diese Methode beginnt mit den üblichen Prüfungen. Danach wird **m_lCurr** auf **NO_POSITION** gestellt, die Hintergrundfarbe der Trennlinie wieder auf den Normalwert eingestellt und mit **IntReposCtrls** (siehe 2.4.2.2.7) alle Controls neu positioniert.

2.4.2.2.6 Public Sub RepositionCtrls()

Parameter: —

Rückgabe: —

Positioniert alle drei Controls und sichert gegen Grenzwertüberschreitungen ab.

Prüft zuerst, ob alle Controls gültig sind, validiert die aktuelle Position und ruft dann die private Methode **IntReposCtrls** (siehe 2.4.2.2.7) auf, um die Controls neu zu positionieren.

2.4.2.2.7 Private Sub IntReposCtrls()

Parameter: —

Rückgabe: —

Positioniert alle drei Controls.

Prüft zuerst, ob alle Controls gültig sind. Falls nicht, wird sofort abgebrochen.

Ansonsten werden die neuen Abmessungen und Positionen der Trennlinie, sowie der beiden verwalteten Controls berechnet und angewandt. Dabei werden die Minimalgrößen (in **m_lMarginLT** und **m_lMarginRB**) beachtet.

---

[23] Hier wird zwar auf den linken geprüft, tatsächlich meldet das System jedoch beim Standardbutton „Links" und beim zweiten Button „Rechts"

[24] Dadurch lässt sich bei versehentlicher Aktivierung der Trennlinie der Vorgang abbrechen, indem man zusätzlich zur linken auch noch die rechte Maustaste drückt.

2.4.2.2.8 Private Function ValidatePos()

Parameter: *ByVal lPos As Long*

Rückgabe: *Long*

Sicherstellen, dass die Position vernünftig ist. Dabei werden die äußeren Grenzen abzüglich der Ränder und Offsets beachtet. Wenn es nicht möglich ist, aus der übergebenen Position `lPos` einen vernünftigen Wert zu berechnen, weil z. B. der Minimalwert größer ist, als der Maximalwert (inkl. aller Ränder), wird `0` zurückgegeben. Ansonsten wird die neue Position innerhalb des gültigen Bereichs ermittelt und als Ergebnis zurückgegeben.

2.4.2.2.9 Private Function AllValid()

Parameter: —

Rückgabe: *Boolean*

Liefert `True`, wenn alle notwendigen Controls richtig angegeben sind, ansonsten `False`.

Prüft, ob alle notwendigen Controls (`m_ctPar`, `m_ctSplit`, `m_ctFirst` und `m_ctSecond`) definiert (nicht `Nothing`) sind. Danach wird getestet, ob das Control `m_ctPar` die Eigenschaft `ScaleWidth` kennt.

Wenn dies alles funktioniert hat, wird noch geprüft, ob die Controls alle unterschiedlich sind. Nur wenn das alles zutrifft, wird `True` zurückgegeben.

2.4.2.3 Attribute

2.4.2.3.1 Margins

Typ: *Long*

Zugriff: *nur Schreiben*

Setzt beide Ränder zwischen den Controls und dem Container. Gelesen werden können die Ränder jedoch nur einzeln (siehe 2.4.2.3.2 und 2.4.2.3.3). Standard ist `DEF_MARGIN` (60 Twips).

2.4.2.3.2 LeftMargin / TopMargin

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Der linke, bzw. obere Rand. Da beide Werte in derselben Variablen gespeichert werden, ist es reine „Kosmetik", dass zwei verschiedene Attribute dafür existieren. Standard ist `DEF_MARGIN` (60 Twips).

2.4.2.3.3 RightMargin / BottomMargin

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Der rechte, bzw. untere Rand. Da beide Werte in derselben Variablen gespeichert werden, ist es reine „Kosmetik", dass zwei verschiedene Attribute dafür existieren. Standard ist `DEF_MARGIN` (60 Twips).

2.4.2.3.4 ActiveColor

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Die Farbe, die die Trennlinie während des Verschiebens annimmt. Standard ist hier die Schattenfarbe der Buttons.

2.4.2.3.5 InactiveColor

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Die Farbe, die die Trennlinie im Ruhezustand annimmt. Standard ist hier die Oberflächenfarbe der Buttons.

2.4.2.3.6 Width / Height

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Die Breite, bzw. Höhe der Trennlinie, je nach dem, ob sie horizontal oder vertikal bewegt werden kann. Da beide Werte in derselben Variablen gespeichert werden, ist es reine „Kosmetik", dass zwei verschiedene Attribute dafür existieren. Standard ist `SPLT_SIZE` (40 Twips).

2.4.2.3.7 FormSize

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Über dieses Attribut kann eine vom Standard abweichende Höhe, bzw. Breite des Elternfensters `m_ctPar` festgelegt werden. Standard ist `INVALID` (-1), so dass `m_ctPar.ScaleWidth`, bzw. `m_ctPar.ScaleHeight` verwendet werden.

2.4.2.3.8 Offset

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Der Abstand der inneren Controls zum Rand von `m_ctPar`. Standard ist `CTRL_OFFSET` (entspricht 20 Twips).

2.4.2.3.9 ShowDirect

Typ: *Boolean*

Zugriff: *Lesen & Schreiben*

Wenn `True`, werden die Controls sofort mitbewegt (Live), ansonsten erst beim Loslassen der Maustaste.

2.4.2.3.10 SplitPos

Typ: *Long*

Zugriff: *Lesen & Schreiben*

Stellt die Position der Trennlinie ein, bzw. gibt sie zurück. Beim Einstellen werden auch alle Controls entsprechend neu positioniert. Beim Lesen wird der obere, bzw. linke Rand der Trennlinie geliefert.

2.5 Fenster und Dialoge – Detail

2.5.1 frmTrend

Dies ist das Hauptfenster der Applikation. Es enthält zum einen das Control **m_Bar** vom Typ **ActiveBar2**, das wiederum das Menü, die Symbol- und die Statusleiste darstellt. Weiterhin sind hier noch zwei Teilfenster, deren Breite sich durch ein Splitterleiste (siehe 2.4.2) gegeneinander verschieben lässt.

Das linke Teilfenster enthält ein Treecontrol, in dem hierarchisch die Verzeichnisse sichtbar sind, in denen wiederum die Trenddateien gespeichert sind. Dieser Verzeichnisbaum liegt auf dem Betriebsserver unter der Freigabe **iTrends**. Der Dateibaum kann auch ausgeblendet werden.

Das rechte Teilfenster beinhaltet das Control **iTrend.ocx**, in dem die Trends dargestellt und bearbeitet werden können.

2.5.1.1 Fenster

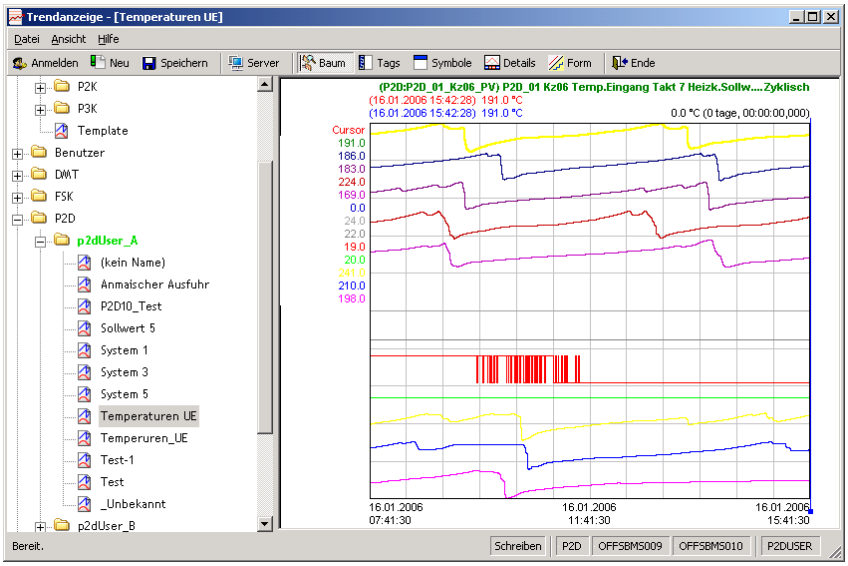


Abbildung 2: frmTrend-Fenster

2.5.1.2 Datenelemente

Name	Typ	Zweck
csReadOnly	Const String	Der Statustext für Trends, die nur gelesen werden können ("Lesen")
csReadWrite	Const String	Der Statustext für Trends, die auch geschrieben werden können ("Schreiben")
m_Split	CSplitter	Die Trennlinie zwischen dem Dateibaum und der Grafikanzeige
m_lSplitPos	Long	Position der Trennlinie (nicht mehr benötigt)
m_lSplitSave	Long	Wenn der Dateibaum ausgeblendet wird, wird hier die aktuelle Position (zum Wiedereinblenden) gespeichert.
m_bReady	Boolean	Wird am Ende der Prozedur Form_Load auf True gesetzt und ermöglicht es, kritische Events bis zu diesem Zeitpunkt zu unterdrücken.
m_bChanged	Boolean	Enthält True, wenn Einstellungen im aktuellen Trend verändert wurden.
m_bSetState	Boolean	Wird dazu verwendet, die Statusanzeige zu aktualisieren.
m_bSelected	Boolean	Dient dazu, nach dem Laden des Dateibaums den Fokus auf das richtige Verzeichnis zu setzen.
m_sCurSource	String	Enthält den Dateinamen des aktuellen Trends auf dem Dateiserver (Original).
m_sCurTarget	String	Enthält den Dateinamen des aktuellen Trends im temporären Verzeichnis (Arbeitskopie).
m_sCurTrgCopy	String	Enthält den Dateinamen der Kopie des aktuellen Trends im temporären Verzeichnis (Vergleichskopie).
m_sRWStatus	String	Enthält csReadOnly bei Nur-Lese-Trends, bzw. csReadWrite bei veränderbaren.
m_SrcNode	Node	Enthält den Verzeichnisknoten, in dem die aktuelle Trenddatei liegt.
m_Legal	Collection	Enthält eine Liste der Verzeichnisse, in den der aktuelle Benutzer Trends speichern darf.

Tabelle 15: frmTrend - Datenelemente

2.5.1.3 Funktionen und Methoden

2.5.1.3.1 Private Sub Form_Load()

Parameter: —

Rückgabe: —

Diese Methode wird vom System aufgerufen, sobald das Fenster aus der Methode Main (siehe 2.3.1.2.1) geladen wird.

Hier wird der Splittermechanismus vorbereitet, indem die entsprechenden Controls eingestellt werden und das Hauptcontrol (m_SplitContainer) dem Control m_Bar als ClientAreaControl zugewiesen wird, wodurch der Container samt den Teilfenstern auf die gesamten Arbeitsfläche ausgedehnt wird.

Im zweiten Schritt wird mit CheckPMS (siehe 2.3.3.2.2) geprüft, ob eine Verbindung zur PMS-Datenbank hergestellt werden kann. Falls nicht, wird eine Warnung ausgegeben und das Programm beendet.

Danach wird mit HookForm (siehe 2.3.5.2.2) das erweiterte Subclassing aktiviert.

Anschließend wird die Statuszeile mit den allgemeinen Informationen eingestellt und mit der Methode ClearTargets (siehe 2.5.1.3.44) evtl. von vorherigen Läufen noch vorhandene Vergleichskopien im temporären Verzeichnis gelöscht.

Nun wird der Splittermechanismus erstellt und eingerichtet. Als nächstes wird, wenn der Parameter -POS angegeben wurde, das Fenster in den gewünschten Bereich des Bildschirms verschoben. Wurde keine gültige Startposition angegeben, holt sich das Programm die letzten Einstellungen aus der Registry und verwendet diese.

Danach wird das Fenster angezeigt[25], mit **InitGUI** der Splitter und die Grundeinstellungen des Trendfensters geladen und angewandt und die Statusleiste erneut aktualisiert, um eventuelle Änderungen darzustellen.

Als nächstes wird der Dateibaum mit der Funktion **FillTree** aufgebaut.

Wenn eine Startdatei angegeben wurde und diese vorhanden ist, wird nun die gesamte Anzeige auf den reinen Betrachter reduziert.

Am Ende wird noch **g_FormLoaded** auf **True** gestellt und die Methode beendet.

2.5.1.3.2 Private Sub Form_Unload()

Parameter: *Cancel As Integer*

Rückgabe: —

Wird vom System aufgerufen, wenn das Fenster entladen wird. Hier wird als erstes das Subclassing mit **UnhookForm** (siehe 2.3.5.2.3) ausgeschaltet. Danach werden mit **ClearTargets** (siehe 2.5.1.3.44) evtl. noch vorhandene Vergleichskopien im temporären Verzeichnis gelöscht.

Wenn beim Aufruf keine Position angegeben wurde, werden jetzt noch verschiedene Einstellungen in der Registry gespeichert.

Danach wird die Verbindung zur Datenbank mit **ClearConn** (siehe 2.3.3.2.4) getrennt und zum Schluss noch mit **UnAuthenticate** (siehe 2.3.1.2.13) die Netzwerkverbindung zum Datenserver aufgehoben.

2.5.1.3.3 Private Sub Form_Resize()

Parameter: —

Rückgabe: —

Wird vom System aufgerufen, wenn die Fenstergröße verändert wurde. Hier wird, wenn der Parameter **-POS** angegeben wurde und die neue Darstellung nicht maximiert ist, das Fenster an seinen durch **-POS** festgelegten Platz gesetzt.

2.5.1.3.4 Public Function Form_Message()

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

Rückgabe: *Boolean*

Dies ist der Callback fürs Subclassing (siehe 2.3.5), in dem die Fensternachricht ausgewertet wird. Dabei wird hier ausschließlich die Nachricht **WM_GETMINMAXINFO** bearbeitet, indem die Minimalgröße des Fensters auf die folgenden Werte festgelegt wird:

Bei positioniertem Fenster: 320 × 200 Pixel

Normalmodus: 640 × 450 Pixel

Dadurch wird vermieden, dass die Fensterelemente unsinnig verschoben oder sogar völlig verdeckt werden. Danach wird der Rückgabewert auf **True** gesetzt, um dem System mitzuteilen, dass die Nachricht behandelt wurde.

Alle anderen Nachrichten werden – durch die Rückgabe von **False** – dem System überlassen.

[25] Der Aufruf von **Show** an dieser Stelle ist eigentlich nicht notwendig, bewirkt hier aber, dass der Bildschirmaufbau etwas flüssiger wirkt.

2.5.1.3.5 Private Sub m_Bar_Resize()

Parameter: *ByVal Left As Long*
ByVal Top As Long
ByVal Width As Long
ByVal Height As Long

Rückgabe: —

Dieses Ereignis wird von **m_Bar** ausgelöst, wenn die Fenstergröße und damit auch die Größe von **m_Bar** verändert wurde. Hier wird – wenn das Fenster nicht minimiert ist – nur die Methode **m_Bar.RecalcLayout** aufgerufen, um die Elemente neu zu positionieren.

2.5.1.3.6 Private Sub m_SplitContainer_Resize()

Parameter: —

Rückgabe: —

Dieses Ereignis wird von **m_SplitContainer** ausgelöst, wenn die Fenstergröße und damit auch die Größe von **m_SplitContainer** verändert wurde. Hier wird – wenn die Trennlinie existiert – nur die Methode **m_Split.RepositionCtrls** aufgerufen, um die beiden Fenster neu zu positionieren.

2.5.1.3.7 Private Sub m_Bar_ToolClick()

Parameter: *ByVal Tool As ActiveBar2LibraryCtl.Tool*

Rückgabe: —

Dieses Ereignis wird von **m_Bar** ausgelöst, wenn einer der Buttons in der Symbolleiste geklickt wurde. Hier wird anhand des Namens des Buttons die entsprechende Funktion oder Methode aufgerufen.

Danach wird mit **SetSymbolState** (siehe 0) die Darstellung der Symbolleiste angepasst[26].

2.5.1.3.8 Private Sub InitGUI()

Parameter: —

Rückgabe: —

Setzt die Position des SplitBars und die Grundeinstellungen des Trendfensters. Diese werden aus der Registry gelesen und dem Trendfenster übergeben.

Wenn dem Benutzer ein Betrieb zugewiesen ist, werden mit **m_Trend.RemoveServer** alle Server aus der internen Liste gelöscht und mit **m_Trend.AddServer** das Trendfenster mit dem zugewiesenen Betrieb verbunden.

Danach wird mit **SetSymbolState** (siehe 0) die Darstellung der Symbolleiste angepasst und das Flag **m_bChanged** gelöscht, um den programminternen Zustand auf „unverändert" zu setzen.

2.5.1.3.9 Private Sub SetSymbolState()

Parameter: —

Rückgabe: —

Setzt den Status der entsprechenden Buttons in der Symbolleiste anhand des Zustands der jeweiligen Eigenschaften.

[26] Um z. B. aktive Modi durch gedrückte Buttons darzustellen.

2.5.1.3.10 Private Sub Relogin()

Parameter: —

Rückgabe: —

Mit dieser Methode kann sich der Benutzer neu am System anmelden. Dabei wird mit der Funktion `Login(True)` (siehe 2.3.1.2.11) das Anmeldefenster angezeigt. Wenn der Benutzer dieses abbricht, wird die Methode verlassen.

Ansonsten wird mit `NewFile` (siehe 0) die Darstellung neu initialisiert und – sofern dem neuen Benutzer ein Betrieb zugewiesen ist – mit `m_Trend.RemoveServer` alle Server aus der internen Liste gelöscht und mit `m_Trend.AddServer` das Trendfenster mit dem neuen Betrieb verbunden.

Zum Abschluss wird der Dateibaum mit der Funktion `FillTree` neu aufgebaut, da sich beispielsweise das Standardverzeichnis geändert haben könnte und die Sichtbarkeit einzelner Verzeichnisse auch vom Benutzer abhängig ist.

2.5.1.3.11 Private Sub ToggleTree()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit des Dateibaums um. Wenn der Dateibaum versteckt ist (durch `m_Split.SplitPos` = 0), wird er auf die in `m_lSplitSave` gemerkte Breite zurückgestellt. Ansonsten wird die aktuelle Breite in `m_lSplitSave` gespeichert und die Breite auf `0` gestellt, wodurch das Fenster völlig verschwindet.

2.5.1.3.12 Private Sub ToggleMenu()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit der Menüleiste um und berechnet mit `m_Bar.RecalcLayout` die Positionierung der Elemente neu.

Da das Menü nicht durch einen Menüeintrag wieder sichtbar gemacht werden kann, wurde dieser Methode der Shortcut `Strg`+`M` zugewiesen.

2.5.1.3.13 Private Sub ToggleSymbolBar()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit der Symbolleiste um und berechnet mit `m_Bar.RecalcLayout` die Positionierung der Elemente neu.

2.5.1.3.14 Private Sub ToggleToolbars()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit der oberen Symbolleisten im Trendcontrol um.

2.5.1.3.15 Private Sub ToggleBrowser()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit des linken Browserfensters im Trendcontrol um.

2.5.1.3.16 Private Sub ToggleDetails()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit des unteren Detailfensters im Trendcontrol um.

2.5.1.3.17 Private Sub ToggleInterpolation()

Parameter: —

Rückgabe: —

Schaltet die Darstellung der Kurven im Trendcontrol zwischen stufig und geglättet um.

2.5.1.3.18 Private Sub ToggleServer()

Parameter: —

Rückgabe: —

Ruft die Methode `ChangeServer` (siehe 2.5.1.3.45) auf.

2.5.1.3.19 Private Sub ToggleStatbar()

Parameter: —

Rückgabe: —

Schaltet die Sichtbarkeit der Statusleiste um und berechnet mit `m_Bar.RecalcLayout` die Positionierung der Elemente neu.

2.5.1.3.20 Private Sub NewFile()

Parameter: —

Rückgabe: —

Initialisiert die Darstellung, indem mit `m_Trend.FileNew` alle Kurven gelöscht werden. Außerdem wird die Hintergrundfarbe des Trendcontrols auf Weiß gestellt und mit `ClearTargets` (siehe 2.5.1.3.44) die temporären Dateien gelöscht.

2.5.1.3.21 Private Sub SaveFile()

Parameter: —

Rückgabe: —

Speichert den aktuellen Trend als `.CRV`-Datei. Hat der aktuelle Trend noch keinen Dateinamen, wird `SaveFileAs` (siehe 2.5.1.3.22) aufgerufen und die Methode beendet.

Anderenfalls wird die Datei mit `m_Trend.FileSave(m_sCurTarget)` im Temporärverzeichnis gespeichert und bei Erfolg `CopyFromTemp` (siehe 2.5.1.3.39) aufgerufen, um die Datei auf den Server zu kopieren.

2.5.1.3.22 Private Sub SaveFileAs()

Parameter: —

Rückgabe: —

Speichert den aktuellen Trend unter einem neuen Namen. Da es je nach Berechtigung des Benutzers nur eine Auswahl an Verzeichnissen gibt, in dem der Trend gespeichert werden darf, musste hier eine eigene Version des Speicher-Dialogs verwendet werden. Dazu wird als erstes mit `GetLegalFolders` (siehe 2.5.1.3.41) eine Liste der erlaubten Verzeichnisse geholt und dem Dialog `dlg` übergeben. Danach wird, wenn bereits ein Dateipfad vorliegt, daraus der reine Dateiname extrahiert und ebenfalls dem Dialog übergeben.

Nun wird der Dialog angezeigt, in dem der Benutzer einen Dateinamen angeben und das Verzeichnis aus der Liste auswählen kann. Wenn der Benutzer diesen Dialog abbricht, liefert er einen Leerstring als Ergebnis zurück und die Methode wird beendet.

Sonst wird sichergestellt, dass der Dateiname mit „**.crv**" endet und – falls die Datei bereits existiert – nachgefragt, ob sie überschrieben werden soll.

Als nächstes wird die Datei mit **m_Trend.FileSave** gespeichert und sofort mit **CopyToTemp** (siehe 0) ins Temporärverzeichnis kopiert. Durch **RefreshTree** (siehe 2.5.1.3.42) mit dem Dateinamen wird der Dateibaum im Verzeichnis der neuen Datei aktualisiert.

Nun wird anstelle der soeben gespeicherten Originaldatei die Arbeitskopie im Temporärverzeichnis geöffnet und der Trend auf Liveanzeige geschaltet.

Abschließend wird der Mauszeiger normalisiert, das Änderungsflag gelöscht und die Statusleiste aktualisiert.

2.5.1.3.23 Private Sub DeleteCurrentFile()

Parameter: —

Rückgabe: —

Dient dazu, die aktuelle Datei zu löschen. Dabei wird die Datei jedoch nicht tatsächlich gelöscht, sondern nur umbenannt, indem an den Dateinamen ein „**x**" angehängt wird. Dadurch wird die Datei für die Routine **WalkTree** (siehe 2.5.1.3.29) ‚unsichtbar'. Der Grund für diese Art des Löschens liegt darin, dass es geplant ist, verschiedene Verzeichnisse zu synchronisieren und es durch die Umbenennung möglich wird, zwischen neuen Dateien auf der einen Seite und gelöschten auf der anderen zu unterscheiden.

Zuerst wird getestet, ob der Benutzer überhaupt das Recht hat, diese Datei zu löschen. Wenn nicht, wird eine Meldung angezeigt und die Routine verlassen.

Ansonsten wird nachgefragt, ob sich der Anwender sicher ist und auch hier beendet, wenn nicht.

Danach wird nachgesehen, ob bereits eine Datei mit dem neuen Dateinamen existiert und diese ggf. gelöscht. Daraufhin wird die Originaldatei umbenannt und die Datei aus dem Dateibaum entfernt, bzw. eine Meldung ausgegeben, falls das Löschen fehlschlug.

2.5.1.3.24 Private Sub m_SplitBar_MouseDown()

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

Rückgabe: —

Dieses Ereignis wird angestoßen, wenn einer der Mausbuttons auf der Trennlinie gedrückt wird. Hier wird nur die Methode **MouseDown** (siehe 2.4.2.2.3) der Klasse **CSplitter** aufgerufen, die alle weiteren Aktionen ausführt.

2.5.1.3.25 Private Sub m_SplitBar_MouseMove()

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

Rückgabe: —

Dieses Ereignis wird angestoßen, wenn die Maus auf der Trennlinie bewegt wird. Hier wird die Methode **MouseMove** (siehe 2.4.2.2.4) der Klasse **CSplitter** aufgerufen, die alle weiteren Aktionen ausführt. Danach wird – falls der linke Mausbutton gedrückt ist – noch **RepositionCtrls** aufgerufen, was aber eigentlich nicht mehr notwendig ist, da **CSplitter** das mit erledigt.

2.5.1.3.26 Private Sub m_SplitBar_MouseUp()

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

Rückgabe: —

Dieses Ereignis wird angestoßen, wenn ein Mausbutton auf der Trennlinie losgelassen wird. Hier wird die Methode **MouseUp** (siehe 2.4.2.2.5) der Klasse **CSplitter** aufgerufen, die alle weiteren Aktionen ausführt. Danach wird noch **RepositionCtrls** aufgerufen (eigentlich nicht mehr notwendig, da **CSplitter** das mit erledigt) und die Position der Trennlinie in **m_lSplitPos** gespeichert.

2.5.1.3.27 Private Function FillTree()

Parameter: —

Rückgabe: *Boolean*

Diese Methode wird aufgerufen, um den Dateibaum (neu) zu füllen. Es wird zuerst die Liste der erlaubten Verzeichnisse neu initialisiert und das Treecontrol geleert.

Anschließend wird mit **GetRoot** (siehe 2.3.1.2.26) das Startverzeichnis ermittelt. Ist dies ein Leerstring, wird die Funktion hier mit **False** abgebrochen.

Nun wird geprüft, ob eine Startdatei angegeben wurde. Wenn ja und die Datei existiert, wird die Funktion jetzt mit **True** beendet.

Ansonsten wird mit **CreateHomeDirIfNeeded** (siehe 2.5.1.3.28) das „eigene" Verzeichnis des Benutzers ermittelt[27] und die rekursive Methode **WalkTree** (siehe 2.5.1.3.29) aufgerufen, die den Dateibaum füllt.

Danach wird der Rückgabewert daraus ermittelt, ob der Baum Verzeichnisse und Dateien enthält und sichergestellt, dass der erste Knoten sichtbar ist.

2.5.1.3.28 Private Function CreateHomeDirIfNeeded()

Parameter: *ByVal sRoot As String*

Rückgabe: *String*

Diese Funktion ermittelt, ob für den aktuellen Benutzer ein eigenes Verzeichnis verwendet und ggf. erstellt werden muss. Wenn dem Benutzer ein Betrieb zugeordnet ist, liefert sie einfach nur einen Leerstring zurück.

Ansonsten wird versucht, im Unterverzeichnis „Benutzer" unter dem Hauptverzeichnis ein Verzeichnis mit dem Anmeldenamen des Benutzers anzulegen. Dieser Pfad wird dann auch zurückgeliefert.

2.5.1.3.29 Private Sub WalkTree()

Parameter: *ByVal sPath As String*
ByVal sMask As String
Optional ByVal sParent As String = ""
Optional ByVal bShow As Boolean = True

Rückgabe: —

Diese Methode sammelt rekursiv alle Unterverzeichnisse und Dateien, die der Maske in **sMask** entsprechen ein und fügt sie in das Treecontrol ein.

---

[27] bzw. erstellt, falls es noch nicht existiert.

In einer statischen Variable **nTreeLevel** wird die aktuelle Tiefe des Baums festgehalten. Sie wird dann auf **0** gesetzt, wenn der Parameter **sParent** ein Leerstring ist[28]. Danach wird ein neues **FileSystemObject** und eine **Collection** erstellt. Der Parameter **sPath** wird normiert (ohne **** am Ende) in **sMyname** gespeichert und der letzte Teil mit **GetFileName** in **sTitle** gespeichert.

Danach wird mit **CalcTag** (siehe 2.5.1.3.30) die aktuelle Berechtigung für diesen Pfad ermittelt und – wenn die Tiefe kleiner als 3 ist – eine Meldung in der Statusleiste angezeigt.

Als nächstes wird der aktuelle Pfad mit dem eigenen Verzeichnis des Benutzers verglichen und die Variablen **bDoExpand** (Wenn **True**, wird der Knoten an dieser Stelle expandiert) und **bAllowed** entsprechend dem Ergebnis und der aktuellen Berechtigung gesetzt.

Wenn dies der erste Aufruf (durch **FillTree**) ist, wird ein Knoten mit dem Text „**Trend-Dateien**" im Baum erzeugt und der Knoten auf jeden Fall expandiert.

Ist es dagegen ein Folgeaufruf, so wird ein Knoten mit dem Text in **sTitle** erzeugt. Expandiert wird dieser Knoten, wenn entweder **bDoExpand True** ist, oder (nur in Ebene 1) entweder das Basisverzeichnis oder der aktuelle Betrieb im Titel steht.

Danach wird die aktuelle Berechtigung im Tag des Knotens gespeichert. Wenn das interne Flag **bAllowed** gesetzt ist, wird der Knoten dann expandiert und selektiert, wenn entweder gar kein eigenes Verzeichnis definiert ist, oder der aktuelle Knoten dem eigenen Verzeichnis entspricht. Außerdem wird bei gesetztem **bAllowed** der aktuelle Pfad in der Collection **m_Legal** gespeichert.

Als nächstes wird mit dem Systemkommando **Dir** in der Collection **cDir** eine Liste aller Unterverzeichnisse des aktuellen Pfads angelegt. Danach wird **nTreeLevel** um eins erhöht, die Verzeichnisliste durch rekursive Aufrufe von **WalkTree** abgearbeitet, **nTreeLevel** wieder auf den vorherigen Wert gesetzt und die Verzeichnisliste gelöscht[29].

Nachdem die Verzeichnisse nun erledigt sind, werden durch ein zweites **Dir**-Kommando alle Dateien des aktuellen Verzeichnisses, die der Maske in **sMask** entsprechen, gesammelt und mit komplettem Pfad und dem reinen Dateinamen als Titel direkt in den Dateibaum eingebaut. Dabei wird jedes Mal noch kontrolliert, ob die Erweiterung auch genau der übergebenen entspricht, da ansonsten auch die durch das zusätzliche „**x**" gelöschte erscheinen würden.

Um das System nicht völlig anzuhalten, wird bei jedem Durchgang oberhalb der dritten Ebene noch **DoEvents** aufgerufen. So können die Statustexte und auch die Veränderungen im Dateibaum schon während des Aufbaus sichtbar werden.

2.5.1.3.30 Private Function CalcTag()

Parameter: *ByVal sPath As String*

Rückgabe: *eSecurity*

Ermittelt die Berechtigung des aktuellen Benutzers im übergebenen Verzeichnis. Dazu wird als erstes ermittelt, ob der Pfad mit „****" beginnt, also ein UNC-Pfad ist. Falls nicht, wird **secNone** zurückgegeben.

Nun wird der Pfad in die einzelnen Teile zerlegt und geprüft, ob der erste Teil nach dem Servernamen „**iTrends**" ist. Falls nicht, wird auch **secNone** zurückgegeben.

Enthält der Pfad nach „**iTrends**" weniger als zwei Elemente, wird auch **secNone** zurückgegeben, da in der ersten Ebene nicht gespeichert werden darf.

Nun wird entschieden, welche Berechtigungen für den aktuellen Pfad zutreffen. Beginnt er mit „**iTrends\Basis**", so dürfen nur Administratoren (Ergebnis = **secAdministrator**) und Power-User (Ergebnis = **secPowerUser**) hier speichern.

---

[28] nur dann, wenn die Methode von **FillTree** aufgerufen wird.

[29] dieser doppelte Durchlauf ist notwendig, da das VB-Kommando **Dir** nicht reentrant-fähig ist.

Der nächste Test greift dann, wenn der zweite Teil des Pfades den aktuellen Betrieb enthält. In diesem Fall wird eine der fünf Möglichkeiten **secUserA**, **secUserB**, **secUserC**, **secUserD** oder **secUserMSR** zurückgegeben.

Ist dies auch nicht der Fall, wird als letztes noch getestet, ob der Pfad dem eigenen Pfad des Benutzers entspricht. Falls ja, wird eine Kombination aus **secUserA**, **secUserB**, **secUserC** und **secUserD** zurückgegeben.

2.5.1.3.31 Private Sub m_Tree_NodeClick()

Parameter: *ByVal Node As MSComctlLib.Node*

Rückgabe: —

Wird aufgerufen, wenn der Benutzer auf einen der Knoten im Dateibaum klickt. Hat **Node** kein Elternelement (Root-Node), so wird nur dafür gesorgt, dass der Knoten expandiert wird.

Sonst wird der **Key**, der dem Pfad entspricht, aus **Node** gelesen und mit dem aktuellen Dateinamen verglichen. Sind die beiden identisch, passiert gar nichts, da die aktuelle Datei nicht nochmals geladen werden muss.

Dann wird getestet, ob der Pfad ein Verzeichnis bezeichnet. Wenn ja, wird nur der Knoten geöffnet und die Methode verlassen.

Danach ist sicher, dass der selektierte Knoten eine Datei enthält. Nun wird der Schreib/Lese-Modus daraus bestimmt, ob die Berechtigung des Benutzers in der Berechtigung enthalten ist, der Mauszeiger auf „Warten" gestellt und mit **ClearTargets** die temporären Dateien gelöscht.

Nun wird die Datei mit **CopyToTemp** ins Temporärverzeichnis kopiert, die Arbeitskopie ins Fenster geladen und Mauszeiger und Statusleiste eingestellt.

2.5.1.3.32 Public Sub SetStatus()

Parameter: *Optional ByVal sStatus As String = "Bereit."*
Optional ByVal bStartTimer As Boolean = False

Rückgabe: —

Setzt die Texte in der Statusleiste.

1. Der übergebene **sStatus** wird dabei in den Bereich ganz links eingetragen.
2. Im zweiten Feld wird der Text **m_sRWStatus** („Lesen" oder „Schreiben") eingetragen.
3. Das dritte Feld erhält den aktuellen Betrieb, bzw. „**---**", falls der Benutzer keinen zugewiesenen Betrieb hat.
4. Hier wird der Name des Datenservers eingetragen.
5. Hier steht der Name des PMS-Datenbankservers.
6. Das letzte Feld wird mit dem vollen Namen des Benutzers gefüllt.

Nun wird mit **m_Bar.RecalcLayout** ein Neuzeichnen der Statusleiste forciert.

Ist der Parameter **bStartTimer** gesetzt, wird der Timer **m_Timer2** auf 5 Sekunden gesetzt und gestartet, um den Statustext nach dieser Zeit wieder zu leeren.

Danach wird noch **SetCaption** (siehe 2.5.1.3.33) aufgerufen.

2.5.1.3.33 Private Sub SetCaption()

Parameter: —

Rückgabe: —

Setzt den Text in der Titelzeile des Fensters. Dieser Text wird zusammengesetzt aus dem Inhalt des Tags dieses Fensters ("**Trendanzeige**") und dem Namen der aktuellen Datei ohne Pfad in **[ ]**. Wurde die Datei verändert, wird außerdem noch ein * angezeigt.

2.5.1.3.34 Private Sub m_Timer2_Timer()

Parameter: —

Rückgabe: —

Stellt die Statusleiste wieder auf die normale Anzeige („Bereit."). Wenn das Flag **m_bSetState** gesetzt ist, wird hier auch **m_bChanged** gelöscht und **SetCaption** (siehe 2.5.1.3.33) aufgerufen.

Anschließend wird der Timer desaktiviert.

2.5.1.3.35 Private Sub SaveToLocal()

Parameter: —

Rückgabe: —

Speichert den aktuellen Trend im temporären Verzeichnis.

Wenn **m_sCurTarget** oder **m_sCurTrgCopy** einen Leerstring enthalten, wird die Methode sofort beendet. Ebenso, wenn das Flag **m_bSetState** gesetzt ist.

Sonst wird die Datei unter dem Namen in **m_sCurTarget** gespeichert und über die Funktion **IsFileChanged** (siehe 2.5.1.3.43) sofort mit dem Inhalt der Vergleichskopie in **m_sCurTrgCopy** verglichen, um das Änderungsflag entsprechend zu setzen. Abschließend wird **SetCaption** (siehe 2.5.1.3.33) aufgerufen, um den Änderungsstatus anzuzeigen.

2.5.1.3.36 Private Sub m_Trend_TagDisplayChanged()

Parameter: *ServerName As String*
TagName As String
ByVal DisplayItem As Long

Rückgabe: —

Wird von **m_Trend** aufgerufen, wenn die Display-Einstellungen geändert wurden. Hier wird einfach **SaveToLocal** (siehe 2.5.1.3.35) aufgerufen, um Änderungen sofort zu ermitteln und in der Titelleiste darzustellen.

2.5.1.3.37 Private Sub m_Trend_TaglistChanged()

Parameter: —

Rückgabe: —

Wird von **m_Trend** aufgerufen, wenn die Tagliste geändert wurde. Hier wird ebenfalls einfach **SaveToLocal** (siehe 2.5.1.3.35) aufgerufen, um Änderungen sofort zu ermitteln und in der Titelleiste darzustellen.

2.5.1.3.38 Private Function CopyToTemp()

Parameter: *ByVal sSrc As String*
Optional ByVal bRW As Boolean = True

Rückgabe: *Boolean*

Mit dieser Funktion wird die Datei in **sSrc** unter dem gleichen Dateinamen in das Temporärverzeichnis kopiert. Weiterhin wird die soeben erstellte Arbeitskopie nochmals als Vergleichskopie (mit der zusätzlichen Erweiterung "**.trd**") ins Temporärverzeichnis kopiert.

Abschließend werden die drei Variablen **m_sCurSource**, **m_sCurTarget** und **m_sCurTrgCopy** auf die entsprechenden Dateipfade gesetzt und **True** als Rückgabewert geliefert.

Sollte beim Kopieren ein Fehler auftreten, werden die drei Variablen geleert, **m_sRWStatus** mit dem Text "**ERR!**" belegt und **False** zurückgegeben.

2.5.1.3.39 Private Function CopyFromTemp()

Parameter: —

Rückgabe: *Boolean*

Diese Funktion kopiert die Arbeitskopie in das entsprechende Serververzeichnis. Wenn eine der drei Variablen **m_sCurSource**, **m_sCurTarget** und **m_sCurTrgCopy** leer ist, wird sofort mit **False** abgebrochen.

Nun wird mit **IsFileChanged** (siehe 2.5.1.3.43) getestet, ob überhaupt Unterschiede zwischen der Arbeits- und der Vergleichskopie bestehen. Wenn nicht, kehrt die Funktion sofort mit **True** zurück, da in diesem Fall kein Kopieren nötig ist.

Sonst wird der Zielpfad mit **GetDefaultPath** (siehe 2.5.1.3.40) ermittelt. Konnte hier kein Pfad ermittelt werden (z. B., weil der Dialog abgebrochen wurde), kehrt die Funktion mit **False** zurück.

Ansonsten wird getestet, ob der Zielpfad und der bisherige Serverpfad ungleich sind und die Datei im Zielpfad bereits existiert. Wenn ja, wird nachgefragt, ob die Datei überschrieben werden soll und, wenn nicht, mit **False** zurückgekehrt.

Danach wird die Datei kopiert und der Rückgabewert auf **True** gesetzt. Sind der Zielpfad und der bisherige Serverpfad ungleich, so wird nun mit **RefreshTree** (siehe 2.5.1.3.42) der Dateibaum aktualisiert und eine Nachricht angezeigt.

2.5.1.3.40 Private Function GetDefaultPath()

Parameter: *ByVal sPath As String*

Rückgabe: *String*

Diese Funktion prüft, ob der aktuelle Benutzer Schreibrechte in dem Verzeichnis hat, in dem die Datei **sPath** liegt. Wenn ja, wird einfach der übergebene Dateiname zurückgeliefert.

Sonst wird geprüft, ob ein eigenes Verzeichnis für den Benutzer existiert. Wenn ja, wird aus diesem Verzeichnis und dem reinen Dateinamen ein neuer Pfad erstellt und zurückgeliefert.

Der nächste Test stellt fest, ob es in der Liste der erlaubten Verzeichnisse (**m_Legal**) nur einen Eintrag gibt. Ist das der Fall, wird ebenfalls dort ein kompletter Dateipfad erstellt und zurückgegeben.

Ansonsten wird ein Dialog aus **dlgSaveAs** (siehe 2.5.6.3.4) erstellt, ihm die Liste der erlaubten Verzeichnisse übergeben und der Dialog angezeigt. Das Ergebnis dieses Dialogs wird als Rückgabewert verwendet[30] und der Dialog zerstört.

2.5.1.3.41 Private Function GetLegalFolders()

Parameter: —

Rückgabe: *Collection*

Liefert eine Collection der Verzeichnisse, in der der aktuelle Benutzer Dateien speichern darf. Diese Collection besteht aus dem eigenen Verzeichnis, sofern vorhanden und der Liste der erlaubten Verzeichnisse in **m_Legal**.

2.5.1.3.42 Private Sub RefreshTree()

Parameter: *ByVal sFile As String*

Rückgabe: —

Diese Methode aktualisiert das Verzeichnis, in dem die Datei in **sFile** liegt, im Dateibaum. Dazu wird zuerst das Verzeichnis aus dem Dateinamen extrahiert und versucht, den entsprechenden Knoten zu finden. Existiert kein solcher Knoten, wird die Methode beendet.

---

[30] Dies kann auch ein Leerstring sein; das muss von der aufrufenden Routine geprüft werden.

Als nächstes werden alle untergeordneten Knoten (= Dateien) des Verzeichnisses nach der Datei durchsucht. Wird die Datei gefunden, wird sie selektiert und die Methode ist beendet. Anderenfalls wird die Datei dem Verzeichnisknoten als neue Datei zugefügt und selektiert.

2.5.1.3.43 Private Function IsFileChanged()

Parameter: *ByVal sFile1 As String, ByVal sFile2 As String*

Rückgabe: *Boolean*

Mit dieser Funktion werden zwei Trenddateien (**sFile1** und **sFile2**) miteinander verglichen. Dabei handelt es sich nicht um einen einfachen Vergleich von Dateigröße oder –datum, sondern der Inhalt wird als Textdatei behandelt und die beiden Texte mit einigen Ausnahmen verglichen.

Dazu wird von beiden Dateien der komplette Inhalt in je eine Stringvariable gelesen. Schlägt das Einlesen einer Datei fehl, so wird in der Fehlerbehandlung **True** (= ungleich) zurückgeliefert.

Nun werden aus beiden Texten die Zeilen entfernt, die mit "**STARTTIME**", bzw. "**ENDTIME**" beginnen, da diese beiden Werte sehr dynamisch sind und sich laufend ändern können, ohne dass ein echter Unterschied im Sinn des Programms besteht.

Danach werden noch alle Leerzeichen und Zeilenwechsel entfernt (da hierbei auch Unterschiede auftreten können, die jedoch nichts an den Eigenschaften der Datei ändern).

Die beiden Ergebnisse werden nun mit der Systemfunktion **StrComp** verglichen und daraus der Rückgabewert (**True** = ungleich, **False** = gleich) ermittelt.

2.5.1.3.44 Private Sub ClearTargets()

Parameter: —

Rückgabe: —

Diese Methode löscht die Dateien, die von **PMS_Trend** im Temporärverzeichnis als Arbeits- und Vergleichskopien (**m_sCurTarget** und **m_sCurTrgCopy**) angelegt wurden und stellt zum Schluss mit **SetCaption** (siehe 2.5.1.3.33) noch die Titelleiste ein.

2.5.1.3.45 Private Sub ChangeServer()

Parameter: —

Rückgabe: —

Mit dieser Methode kann zwischen den beiden Betriebsservern als Datenlieferanten umgeschaltet werden.

Wenn der aktuelle Benutzer keinen zugewiesenen Betrieb hat, wird die Methode sofort beendet, da in diesem Fall die Daten von einem zentralen Server geliefert werden und ein Umschalten nicht möglich ist.

Ansonsten wird ein Objekt **oAlias** vom Typ **UISupport.dbAlias**[31] erstellt und dieses auf TCP/IP eingestellt. Danach werden aus der PMS-Tabelle **[wwAlias]** die Namen der beiden Datenserver ermittelt und mit der Methode **oAlias.MachineName** der jeweils andere aktiviert. Danach wird noch **oAlias.Update** aufgerufen, um die Änderungen zu übernehmen.

Zum Abschluss wird noch dem Trendcontrol der jetzt aktuelle Server zugefügt, die Statusleiste angepasst und das Trendcontrol auf Livemodus gestellt.

[31] gehört zum Trendcontrol von ActiveFactory.

2.5.2 dlgAbout

Dieser Dialog dient nur dazu, die Version sowohl des Programms, als auch der verwendeten Controls anzuzeigen.

2.5.2.1 Dialogfenster

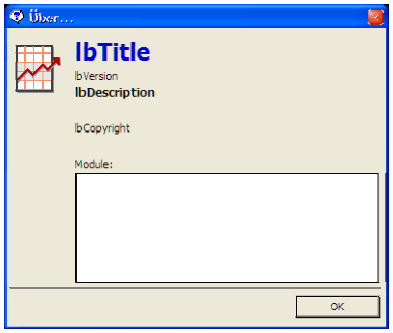


Abbildung 3: dlgAbout-Fenster

2.5.2.2 Datenelemente

Dieser Dialog enthält keine Datenelemente.

2.5.2.3 Funktionen und Methoden

2.5.2.3.1 Private Sub Form_Load()

Parameter: —

Rückgabe: —

Dieses Ereignis wird beim Laden des Dialogs aufgerufen.

Hierbei werden der Fenstertitel und die Überschrift (im Bild „`lbTitle`“) auf den Programmnamen eingestellt[32]. Danach werden noch die Controls `lbVersion`, `lbDescription` und `lbCopyright` auf die Werte aus den Dateiinformationen gesetzt. Als letztes werden in der Textbox `txModules` noch die Versionen der folgenden Controls eingetragen:

- `SSLibUI.StdDlg`
- `DDActiveReports2.ActiveReport`
- `TrueOleDBGrid70.TDBGrid`
- `MSFlexGridLib.MSFlexGrid`
- `ADODB.Command`
- `ADODB.Connection`
- `OLEDB`
- `ODBC`

Die Versionen dieser Controls und Referenzen werden mit `GetVerString` (siehe 2.3.1.2.2) ermittelt. Dieser Dialog wird in nahezu unveränderter Form in allen PMS-Programmen verwendet, daher darf es nicht verwundern, dass einige der hier aufgelisteten Controls überhaupt nicht vorhanden sind.

2.5.2.3.2 Private Sub btOK_Click()

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der **OK**–Button gedrückt wird und entlädt den Dialog.

---

[32] Die Überschrift wird an die drei Controls `lbTitle(0…2)` übergeben. Diese werden dann so angeordnet, dass sich ein 3D-Effekt ergibt.

2.5.3 dlgErrHndl

Dieser Dialog wird von der Methode **ErrorHandler** (siehe 2.3.2.2.1) angezeigt, wenn ein schwerer oder nicht behandelter Fehler aufgetreten ist. Die Informationen dazu werden dabei vor dem Anzeigen über die Attribute (siehe 2.5.3.4) eingestellt.

2.5.3.1 Dialogfenster

Abbildung 4: dlgErrHndl-Fenster

2.5.3.2 Datenelemente

Name	Typ	Zweck
m_sFullErrorInfo	String	Enthält den kompletten Fehlertext
m_sUserName	String	Enthält den Anmeldenamen des aktuellen Benutzers
m_sUserFullName	String	Enthält den kompletten Namen des aktuellen Benutzers
m_sMailReceiver	String	Enthält die Mailadresse des Empfängers
m_sSMTPServer	String	Enthält den Namen des SMTP-Servers
sHtmlHead	Const String	Der Kopfbereich der HTML-Datei
sHtmlBody	Const String	Der Textbereich der HTML-Datei, mit Benutzernamen
sHtmlBody2	Const String	Der Textbereich der HTML-Datei, ohne Benutzernamen
sHtmlFoot	Const String	Der Endbereich der HTML-Datei

Tabelle 16: dlgErrHndl - Datenelemente

2.5.3.3 Funktionen und Methoden

2.5.3.3.1 Private Sub Form_Activate()

Parameter: —

Rückgabe: —

Wird aufgerufen, sobald das Dialogfenster aktiviert wird. Dabei werden die vorhandenen Fehlertexte in den Controls entsprechend angeordnet und der Button **Mail senden…** versteckt, wenn kein Mailempfänger definiert ist.

2.5.3.3.2 Private Sub btClose_Click()

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der **Schließen**–Button gedrückt wird und versteckt den Dialog.

2.5.3.3.3 *Private Sub btProcessError_Click()*

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der **Mail senden…** –Button gedrückt wird und öffnet den unteren Teil des Dialogs, in dem noch ein Kommentar eingegeben und eine Mail gesendet werden kann.

2.5.3.3.4 *Private Sub btEMail_Click()*

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der **Senden** –Button gedrückt wird. Wurde noch keine Beschreibung eingegeben, so wird nur eine Aufforderung angezeigt und die Methode beendet.

Sonst wird ein Objekt **Msg** vom Typ **CDO.Message** erstellt, über das eine Mail erstellt wird. Wenn kein SMTP-Server festgelegt ist, wird als Standard "**OFFSBNT001.EMR-OFF.LOCAL**" verwendet.

Nun werden noch einige Einstellungen gesetzt und als Betreffzeile der Programmname und "**Fehlermeldung**" gesetzt. Der Nachrichtentext wird mit der Funktion **ErrorHTML** (siehe 2.5.3.3.6) erstellt.

Nun wird das mittlere Control mit dem Text "Nachricht wird gesendet" sichtbar gemacht, der Cursor auf Warten gestellt und die Mail gesendet.

Zum Abschluss wird das mittlere Fenster wieder versteckt, der Cursor wieder zurückgestellt, das Objekt **Msg** zerstört und das gesamte Fenster wieder versteckt.

2.5.3.3.5 *Private Sub txUserDescription_GotFocus()*

Parameter: —

Rückgabe: —

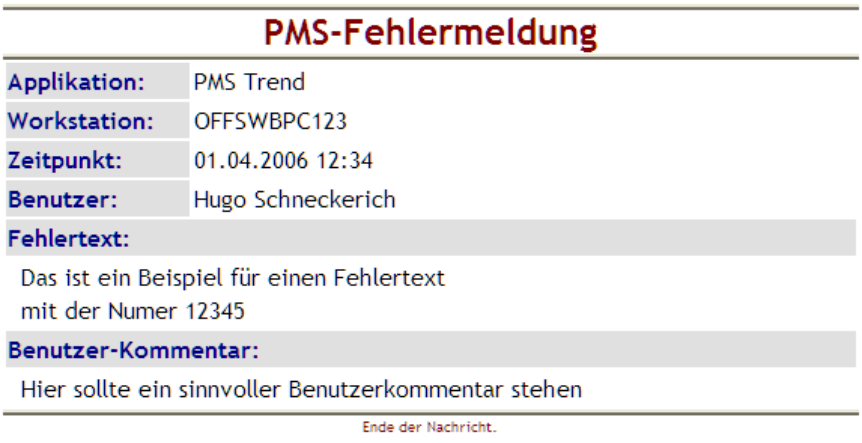
Wird aufgerufen, wenn das Beschreibungsfeld im unteren Bereich den Fokus erhält. Hier wird einfach der gesamte Text markiert, so dass der Benutzer evtl. bestehenden Text direkt überschreiben kann.

2.5.3.3.6 *Private Function ErrorHTML()*

Parameter: —

Rückgabe: *String*

Erstellt aus den verschiedenen Informationen ein tabellarisch aufgebautes HTML-Dokument, das dann als **HTMLBody** verwendet wird. Ein Beispiel für eine solche HTML-Meldung:



PMS-Fehlermeldung

Applikation:	PMS Trend
Workstation:	OFFSWBPC123
Zeitpunkt:	01.04.2006 12:34
Benutzer:	Hugo Schneckerich

Fehlertext:

Das ist ein Beispiel für einen Fehlertext
mit der Numer 12345

Benutzer-Kommentar:

Hier sollte ein sinnvoller Benutzerkommentar stehen

Ende der Nachricht.

Abbildung 5: HTML-Fehlermeldung

2.5.3.3.7 Private Function ErrorText()

Parameter: —

Rückgabe: *String*

Setzt einen einigermaßen formatierten (ASCII-)Text aus den verschiedenen Informationen zusammen, der dann als `TextBody` verwendet werden kann.

Anmerkung: Wird nicht mehr verwendet.

2.5.3.4 Attribute

2.5.3.4.1 FullErrorInfo

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt den Fehlertext, der angezeigt und / oder versendet wird.

2.5.3.4.2 Description

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt den Beschreibungstext.

2.5.3.4.3 Kontext

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt den Kontext, in dem der Fehler auftrat.

2.5.3.4.4 ADOError

Typ: *String*

Zugriff: *Lesen und Schreiben*

Setzt oder liest die Fehlerbeschreibung für Datenbankfehler (zusätzlich zu `FullErrorInfo`).

2.5.3.4.5 UserName

Typ: *String*

Zugriff: *Lesen und Schreiben*

Setzt oder liest den Anmeldenamen des Benutzers.

2.5.3.4.6 FullName

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt oder liest den kompletten Namen des Benutzers.

2.5.3.4.7 Receiver

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt die Emailadresse des Empfängers[33].

[33] Diese sollte in der Registry eingetragen sein

2.5.3.4.8 SMTPServer

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt den SMTP-Server, über den die Mail versendet wird. Ist der Wert nicht definiert, so wird als Standard "`OFFSBNT001.EMR-OFF.LOCAL`" verwendet.

2.5.4 dlgGetUser

Dieser Dialog dient zur Eingabe eines Benutzernamens mit mehr Rechten, als der angemeldete Benutzer hat. Er wird von der Funktion `SingleItemAllowed` (siehe 2.4.1.2.8) angezeigt, wenn nötig.

Hier kann dann der Anmeldename und das Kennwort eines Benutzers mit höheren Rechten eingegeben werden, so dass die Funktion, die diese Rechte benötigt, ausgeführt werden kann.

2.5.4.1 Dialogfenster

Abbildung 6: dlgGetUser-Fenster

2.5.4.2 Datenelemente

Name	Typ	Zweck
LoginSucceeded	Boolean	Enthält `True`, wenn die Eingabe gültig war, sonst `False`.
sSqlCheckUserItem	Const String	SQL-Abfrage zur Kontrolle der Eingabe.
m_sItem	String	Die Bezeichnung der Funktion, die erlaubt werden soll.

Tabelle 17: dlgGetUser - Datenelemente

2.5.4.3 Funktionen und Methoden

2.5.4.3.1 Private Sub Form_Load()

Parameter: —

Rückgabe: —

Wird beim Laden des Dialogs aufgerufen. Hier wird der Inhalt des Kennwort-Feldes gelöscht.

2.5.4.3.2 Private Sub Form_Activate()

Parameter: —

Rückgabe: —

Wird beim Aktivieren des Dialogs aufgerufen. Wenn ein Benutzername bereits eingetragen ist, wird der Fokus stattdessen auf das Kennwortfeld gesetzt.

2.5.4.3.3 *Private Sub m_OK_Click()*

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der **OK**–Button geklickt wird. Mit der Funktion **CheckPwd** (siehe 2.5.4.3.10) wird geprüft, ob Benutzername und Kennwort gültig sind und ausreichende Rechte für die gewünschte Funktion besitzen. Wenn ja, wird **LoginSucceeded** auf **True** gesetzt und der Dialog versteckt.

Wenn nicht, wird eine entsprechende Nachricht angezeigt und der Fokus ins Kennwortfeld gesetzt.

2.5.4.3.4 *Private Sub m_Cancel_Click()*

Parameter: —

Rückgabe: —

Setzt **LoginSucceeded** auf **False** und versteckt den Dialog.

2.5.4.3.5 *Private Sub m_User_GotFocus()*

Parameter: —

Rückgabe: —

Selektiert den kompletten Text im Benutzernamen, sobald das Feld den Fokus erhält.

2.5.4.3.6 *Private Sub m_User_Change()*

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der Text im Benutzernamen geändert wird. Ruft wiederum **CheckOK** (siehe 2.5.4.3.9) auf.

2.5.4.3.7 *Private Sub m_Pwd_GotFocus()*

Parameter: —

Rückgabe: —

Selektiert den kompletten Text im Kennwortfeld, sobald das Feld den Fokus erhält.

2.5.4.3.8 *Private Sub m_Pwd_Change()*

Parameter: —

Rückgabe: —

Wird aufgerufen, wenn der Text im Kennwort geändert wird. Ruft wiederum **CheckOK** (siehe 2.5.4.3.9) auf.

2.5.4.3.9 *Private Sub CheckOK()*

Parameter: —

Rückgabe: —

Wird von **m_User_Change** (siehe 2.5.4.3.6) und **m_Pwd_Change** (siehe 2.5.4.3.8) aufgerufen. Hier wird nur geprüft, ob sowohl die Funktionsbeschreibung, als auch der Benutzername nicht leer sind. Wenn ja, wird der **OK**–Button aktiviert, sonst nicht.

2.5.4.3.10 Private Function CheckPwd()

Parameter: *ByVal sItem As String*
ByVal sUser As String
ByVal sPwd As String

Rückgabe: *Boolean*

Diese Funktion testet mit dem SQL-Kommando **sSqlCheckUserItem**, ob die Funktion **sItem** mit dem Benutzer **sUser** und dem Kennwort **sPwd** ausgeführt werden darf und liefert entsprechend **True** oder **False** zurück.

2.5.4.4 Attribute

2.5.4.4.1 ItemName

Typ: *String*

Zugriff: *Nur Schreiben*

Setzt die Bezeichnung der gewünschten Funktion.

2.5.4.4.2 User

Typ: *String*

Zugriff: *Lesen und Schreiben*

Setzt den Benutzernamen. Dieser kann somit vorbelegt werden, um bei mehrfachen Aufrufen nur noch das Kennwort eingeben zu müssen.

2.5.5 dlgLogin

Wird zu Beginn des Programms aufgerufen, um den Benutzernamen und das Kennwort eingeben zu lassen.

2.5.5.1 Dialogfenster

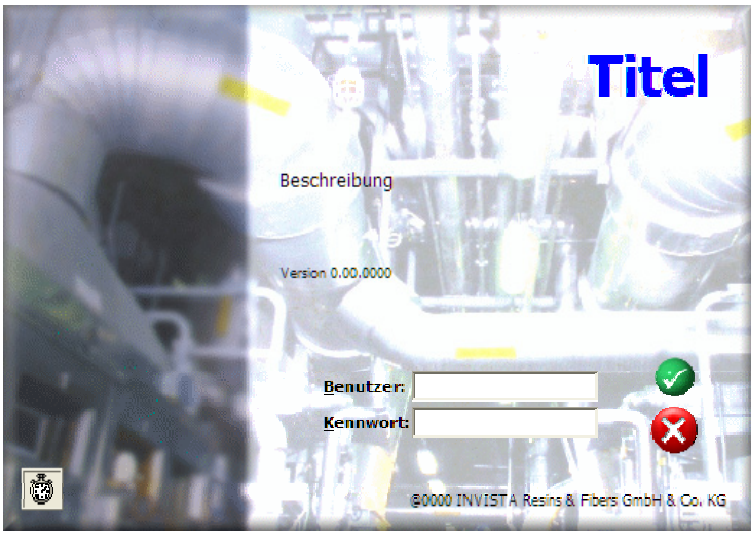


Abbildung 7: dlgLogin-Fenster

2.5.5.2 Datenelemente

Name	Typ	Zweck
ReleaseCapture	Function	
SendMessage	Function	
HTCAPTION	Const Long	
WM_NCLBUTTONDOWN	Const Long	
Cancel	Boolean	
m_sUser	String	
m_sUserFullName	String	
m_sPassword	String	
m_bRelogin	Boolean	

Tabelle 18: dlgLogin - Datenelemente

2.5.5.3 Funktionen und Methoden

2.5.5.3.1 Private Sub Form_Load()

Parameter: —

Rückgabe: —

2.5.5.3.2 Private Sub Form_KeyUp()

Parameter: *KeyCode As Integer*
Shift As Integer

Rückgabe: —

2.5.5.3.3 Private Sub btOK_Click()

Parameter: —

Rückgabe: —

2.5.5.3.4 Private Sub btCancel_Click()

Parameter: —

Rückgabe: —

2.5.5.3.5 Private Sub imgLogo_MouseDown()

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

Rückgabe: —

2.5.5.3.6 Private Sub lbl_MouseDown()

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

Rückgabe: *—*

2.5.5.3.7 Private Sub MoveWindow()

Parameter: *—*

Rückgabe: *—*

2.5.5.3.8 Private Sub timDBCon_Timer()

Parameter: *—*

Rückgabe: *—*

2.5.5.3.9 Private Sub txtUser_GotFocus()

Parameter: *—*

Rückgabe: *—*

2.5.5.3.10 Private Sub txtPwd_GotFocus()

Parameter: *—*

Rückgabe: *—*

2.5.5.3.11 Private Sub txtUser_KeyPress()

Parameter: *KeyAscii As Integer*

Rückgabe: *—*

2.5.5.3.12 Private Sub txtPwd_KeyPress()

Parameter: *KeyAscii As Integer*

Rückgabe: *—*

2.5.5.3.13 Private Function CheckUserPwd()

Parameter: *—*

Rückgabe: *Boolean*

2.5.5.4 Attribute

2.5.5.4.1 User

Typ: *String*

Zugriff: *Lesen und Schreiben*

2.5.5.4.2 UserFullName

Typ: *String*

Zugriff: *Lesen und Schreiben*

2.5.5.4.3 Password

Typ: *String*

Zugriff: *Lesen und Schreiben*

2.5.5.4.4 Relogin

Typ: *Boolean*

Zugriff: *Lesen und Schreiben*

2.5.6 dlgOptions

Mit diesem Dialog können die Zugriffsdaten für die PMS-Datenbank eingestellt werden.

2.5.6.1 Dialogfenster

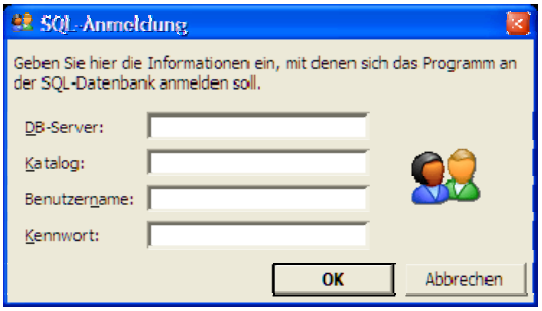


Abbildung 8: dlgOptions-Fenster

2.5.6.2 Datenelemente

Name	Typ	Zweck
m_sDBServer	String	
m_sDbCatalog	String	
m_sDbUser	String	
m_sDbPwd	String	
Cancelled	Boolean	

Tabelle 19: dlgOptions - Datenelemente

2.5.6.3 Funktionen und Methoden

2.5.6.3.1 Private Sub Form_Load()

Parameter: —

Rückgabe: —

2.5.6.3.2 Private Sub cmdOK_Click()

Parameter: —

Rückgabe: —

2.5.6.3.3 Private Sub cmdCancel_Click()

Parameter: —

Rückgabe: —

2.5.6.3.4 Private Sub txtDbServer_GotFocus()

Parameter: —

Rückgabe: —

2.5.6.3.5 Private Sub txtDbServer_KeyPress()

Parameter: *KeyAscii As Integer*

Rückgabe: —

2.5.6.3.6 Private Sub txtCatalog_GotFocus()

Parameter: —

Rückgabe: —

2.5.6.3.7 Private Sub txtCatalog_KeyPress()

Parameter: *KeyAscii As Integer*

Rückgabe: —

2.5.6.3.8 Private Sub txtUsername_GotFocus()

Parameter: —

Rückgabe: —

2.5.6.3.9 Private Sub txtPwd_GotFocus()

Parameter: —

Rückgabe: —

2.5.6.4 Attribute

2.5.6.4.1 ConnectionString

Typ: String

Zugriff: Nur Lesen

2.5.7 dlgSaveAs

Wird angezeigt, wenn ein Dateiname und / oder ein Pfad zum Speichern ausgewählt werden muss. In der Pfadliste werden dabei nur die Pfade angezeigt, in denen dem Benutzer das Schreiben erlaubt ist.

2.5.7.1 Dialogfenster

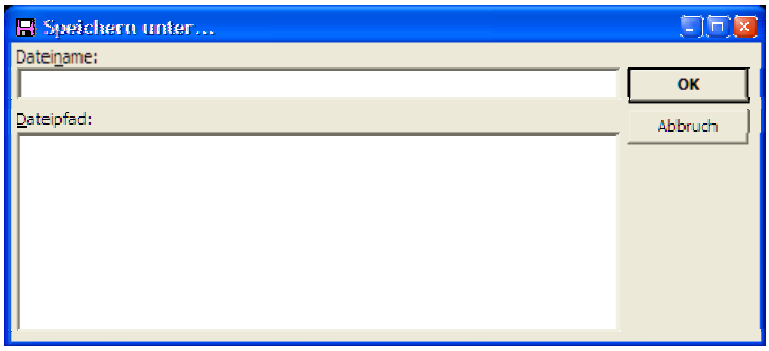


Abbildung 9: dlgSaveAs-Fenster

2.5.7.2 Datenelemente

Name	Typ	Zweck
m_sRoot	String	
m_sResult	String	
m_sFilename	String	
m_sFolders	Collection	

Tabelle 20: dlgSaveAs - Datenelemente

2.5.7.3 Funktionen und Methoden

2.5.7.3.1 Private Sub Form_Initialize()

Parameter: —

Rückgabe: —

2.5.7.3.2 Private Sub Form_Load()

Parameter: —

Rückgabe: —

2.5.7.3.3 Private Sub Form_Unload()

Parameter: *Cancel As Integer*

Rückgabe: *—*

2.5.7.3.4 Public Function Form_Message()

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

Rückgabe: *Boolean*

2.5.7.3.5 Private Sub Form_Resize()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.6 Public Sub AddFolder()

Parameter: *ByVal s As String*

Rückgabe: *—*

2.5.7.3.7 Private Sub m_btCancel_Click()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.8 Private Sub m_btOK_Click()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.9 Private Sub m_lsTarget_Click()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.10 Private Sub m_lsTarget_DblClick()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.11 Private Sub m_txFilename_Change()

Parameter: *—*

Rückgabe: *—*

2.5.7.3.12 Private Sub CheckOK()

Parameter: —

Rückgabe: —

2.5.7.3.13 Private Sub m_txFilename_GotFocus()

Parameter: —

Rückgabe: —

2.5.7.4 Attribute

2.5.7.4.1 Filename

Typ: String

Zugriff: Lesen und Schreiben

2.5.7.4.2 Result

Typ: String

Zugriff: Nur Lesen