Sage 50 Rechnungswesen Extra/Version 2019

OLE SDK (SOK)

Inhaltsverzeichnis

1.0 SOK Programmierung

Dieser Teil der Dokumentation gibt Ihnen eine kleine Einführung in die Programmierung von Sage 50 OLE Objekten.

1.1 Einführung

Mit Hilfe des SOK können Sie die Sage 50 Objekte und Methoden in Ihren Programmen verwenden und eine nahtlose Integration der Sage 50 Produkte erreichen. Im SOK finden Sie Dokumentation, Hilfsmittel und Beispiele für die Programmierung mit den SOK Objekten.

Die meisten Beispiele in dieser Dokumentation wurden mit der englischen «Visual Basic Professional Edition» geschrieben und getestet. Das ist jedoch nicht zwingend, auch ein deutsches Visual Basic, Excel oder Word kann benützt werden. Sie müssten einfach die entsprechenden, deutschen Befehle der BASIC-Sprache einsetzen.

1.2 Installation

SOK verwendet einige Komponenten vom Sage 50 Rechnungswesen (REWE). Es ist deshalb zwingend, dass REWE vorher richtig installiert wurde.

1.2.1 Das Erste Objekt

Der einfachste Weg die Funktionalität von SOK zu testen, ist die folgenden Zeilen mit Visual Basic oder Excel zu erfassen und durchzuführen:

```
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")
Man.Login (1)
```

Wenn Sie eine «Mandant öffnen»-Dialogbox vor sich haben, ist alles Startbereit. Ein Mandant-Objekt wurde erstellt, und die Mandant.Login Funktion, die nach einem Sage50 Mandant fragt, wurde ausgeführt.

1.2.2 Fehlerquellen

Wenn Sie eine Fehler-Meldung erhalten im Sinne von: «OLE Automation Server kann das Objekt nicht erstellen.» überprüfen Sie folgendes:

Registry Eintrag

FibuSDK muss in der „System Registry“ bzw. „Windows Registry“ eingetragen sein. Um das zu überprüfen, starten Sie den Registry Editor (regedit.exe) und Suchen nach dem Text «FibuNT.Mandant».

Wenn Sie keinen Registry-Eintrag mit dem gesuchen Text finden, so können Sie versuchen, FibuSDK nachträglich zu registrieren. Öffnen Sie dazu ein Konsolen-Fenster (cmd.exe) mit Administrator-Rechten und geben Sie folgende Kommandos ein:

> *32-Bit Systeme:* cd "%ProgramFiles%\Sage\Sage50\Prog"
> *64-Bit Systeme:* cd "%ProgramFiles(x86)%\Sage\Sage50\Prog"
> regsvr32.exe fbsdk50.dll

Die notwendigen Einträge werden jetzt automatisch in die Registry geschrieben.

BTRIEVE

Versuchen Sie mit REWE einen Btrieve-Mandanten zu öffnen. Wenn das geht, ist Btrieve richtig installiert.

1.3 Funktionsweise

SOK basiert auf der «OLE 2 Automation» Technik von Microsoft. Im Wesentlichen arbeiten Sie mit OLE Objekten, die bestimmte Eigenschaften und Methoden (Funktionen) aufweisen, durch die Sie Ihre Arbeit erledigen können. Die Objekthierarchie von SOK ist sehr flach. Sie erstellen ein Mandant-Objekt und mit diesem erstellen Sie alle anderen Objekte.

1.3.1 Ein neues Fibu-Konto

Damit Sie Sage 50 Daten bearbeiten können, zum Beispiel das Fibu- Konto 1000, gehen Sie folgendermassen vor:

1. Ein **Mandant**-Objekt mit der OLE-Kreationsfunktion der jeweiligen Programmiersprache erstellen
   (bei Visual Basic z.B. **CreateObject**).
   – Mit diesem Schritt wird SOK inkl. zugehörigen DLLs geladen.
2. Die **Login**-Funktion des **Mandant**-Objekts aufrufen.
   – Ein REWE Mandant wird geöffnet.
3. Die **NeuKonto**-Funktion des **Mandant**-Objekts aufrufen.
   – Sie erhalten ein Konto-Objekt mit dem Sie Fibu-Konten lesen und schreiben können.
4. Die **Lesen**-Funktion des **Konto**-Objekts mit dem Parameter «1000» aufrufen.
   – Das Konto-Objekt enthält jetzt das Fibu-Konto 1000. Sie können jetzt alle Eigenschaften des Kontos 1000 ansehen und einige auch ändern.
5. Ändern Sie die Eigenschaft KontoNr auf eine neue Konto-Nr und rufen die Funktion Einfuegen des Konto-Objekts auf.
   – Schon haben Sie ein neues Fibu-Konto erstellt.
6. Das **Konto Objekt**, das in Schritt 3 erstellt wurde **zerstören**
7. Die **Logout**-Funktion des **Mandant**-Objekts aufrufen
8. Das **Mandant Objekt**, das in Schritt 1 erstellt wurde **zerstören**

1.3.2 Mehrere Mandanten

Im gleichen Programm können auch mehrere Mandanten gleichzeitig geöffnet werden. Sie können dann zum Beispiel Datenvergleiche mit dem Vorjahr machen.

WICHTIG: Zerstörung der Objekte die über das Mandant Objekt erstellt werden

Über das Mandant Objekt können verschiedene Objekte erzeugt werden. So z.B. kann ein neues Konto Objekt via Funktion Mandant.NeuKonto erzeugt werden.

Solche Objekte sollten **vor dem Aufruf der Funktion Mandant.Logout** ordnungsgemäss zerstört werden, damit der allozierte Speicherbereich im Arbeitsspeicher wieder freigegeben wird. Geschieht das nicht, kann es zu unkontrolliertem Verhalten kommen, wodurch Clients die FibuSDK verwenden, abstürzen können.

Beispiel:

```
Dim Mandant As Object
```

```
Dim Buchung As Object
Dim Konto As Object
Dim Steuer As Object

Dim MandantenPfad As String
Dim LoginType As Integer

' Mandanten Objekt erzeugen und Mandant öffnen
Set MandantenPfad = "Pfad/zum/Mandanten/Verzeichnis"
Set LoginType = 1 'FibuNT Dateien d.h. z.B. Buchung, Konto, Steuer usw.
Set Mandant = CreateObject("FibuNT.Mandant")
Mandant.Login(loginType, MandantenPfad)

' Buchung-, Konto- und Steuer-Objekt erzeugen
Set Buchung = Mandant.NeuBuchung()
Set Konto = Mandant.NeuKonto()
Set Steuer = Mandant.NeuSteuer()

[ … ]

' Objekte zerstören
Set Buchung = Nothing
Set Konto = Nothing
Set Steuer = Nothing

' Mandant schliessen
Mandant.Logout()

' Mandant-Objekt zerstören
Set Mandant = Nothing
```

1.4 Entwicklungsumgebung

1.4.1 Visual Basic

Visual Basic Programmierer haben es in diesem Fall wieder einfach, denn die meisten Beispiele in diesem Dokument stammen aus Visual Basic.

1.4.2 Visual C++

Für C++-Programmierer steht eine «Automation Type Library» zur Verfügung mit der Sie ein neues C++-Objekt erstellen können, über das Sie auf die SOK Objekte zugreifen. Das C++-Objekt kann Ihnen, ab MS- VC++ Version 1.5, der Class-Wizard automatisch erstellen.

Es ist wichtig, nach Erhalt einer neuen Version von SOK, das C++ Objekt neu zu erstellen, damit alle Verweise auf die Eigenschaften und Methoden der SOK Objekte stimmen.

1.4.3 Visual C++ 6.0

Ab Version 6.0 von Visual C++ können die Compiler unterstützten «smart pointers» verwendet wenden. Folgende Zeilen öffnen die Mandant- Loginbox, danach wird der Konto-Browser gezeigt:

```
#import "fbsdk50.dll"
using namespace fibusdk;
...
MandantPtr pMan ("FibuNT.Mandant");
pMan->Login (1, "", "");
KontoPtr pKonto = pMan->NeuKonto();
pKonto->Browser2 ("2000", vtMissing);
```

Vergessen Sie nicht **CoInitialize** oder **OleInitialize** vorher aufzurufen, damit OLE initialisiert wird. Auch müssen sie **try/catch** verwenden um Fehler zu behandeln.

1.4.4 Microsoft Access, Excel, PowerPoint und Word

Diese Produkte haben in den neueren Versionen das Visual Basic for Applications voll eingebaut, so dass die Bearbeitung von SOK Objekte fast genau gleich ist wie bei Visual Basic.

1.4.5 Delphi 2

In Dephi 2 (32Bit) ist die Erstellung eines OLE-Objects sehr einfach. Man importiert die Unit **OLEAuto** und kann dann mit dem Befehl **CreateOleObject** ein OLEObject erzeugen und einer Variablen vom Typ **Variant** zuordnen.

Zum Beispiel:

```
uses OleAuto;
..
var Man, Struk: Variant;

Man := CreateOleObject( 'FibuNT.Mandant' );
Man.Login( 1 , 'c:\ProgramData\Sage\Data\Rewe\SageDemo16' );

Struk := Mandant.NeuStruktur;
kasse := Struk.Lesen2( '1000' , 3 );
```

1.4.6 Delphi 1

In Delphi 1 (16Bit) ist die Sache wesentlich mühsamer und nur mit einer zugekauften third party Units zu machen. Hat man diese Unit gekauft und importiert, muss man als erstes alle zu verwendenden OLE-Objekte als Pascal-Klassen nachbilden deren Methoden und Properties schliesslich auf das richtige OLE-Objekt zugreifen.

Zum Beispiel:

```
-----------------------------------
interface;
uses OleAuto; // zugekaufte Unit
...
type
TMandant = class(TOleObject)
public
Konto:TKonto;
function Login (i:integer; pfad:string; pswrd:string) : integer;
function NeuKonto:TKonto;
end;

-----------------------------------
implementation;
function TMandant.Login (i:integer; pfad:string; pswrd:string):
integer;
var j:integer;
s:array [1..260] of Char;
ps:PChar;
begin
ps := addr(s);
StrPCopy (addr(s), pswrd);
SetOleMethodArg('PChar',ps);
SetOleMethodArg('Integer',i);
CallOlefunction('Login','Integer',j);
result:=j;
end;
```

Aufruf:

```
var Mandant : TMandant;
Mandant := TMandant.CreateObject('FibuNT.Mandant');
Mandant.Login( 1 , 'C:\ProgramData\Sage\Data\Rewe\SageDemo16'," )
```

1.4.7 Andere

Inzwischen gibt es schon viele Entwicklungsumgebungen mit denen OLE Automation Objekte erstellt und bearbeitet werden können. SOK wurde nach strengen Richtlinien programmiert und sollte sich deshalb auch in andere Umgebungen nahtlos integrieren lassen.

1.5 Die Type Library

Zweck

FibuSDK stellt eine Type Library zur Verfügung, in der die Namen von Klassen, Schnittstellen, Funktionen und Properties sowie die Typen der Übergabeparameter von Funktionen deklariert sind. Die Verwendung der Type Library erleichtert das Programmieren mit FibuSDK, da man von der Entwicklungsumgebung beim Schreiben sofort Vorschläge erhält, welche Namen und Parameter es gibt. Man muss dadurch weniger nachschlagen und macht auch weniger Syntax-Fehler, da die richtige Schreibweise sofort vorgegeben wird.

Je nach Programmiersprache und Entwicklungsumgebung muss eine Type Library zwingend eingebunden werden (C++), ist unter Umständen aber auch nur optional oder wird gar nicht benötigt (VBA). Grundsätzlich muss der Entwickler selber entscheiden, ob die Type Library in der Entwicklungsumgebung benötigt wird, weiter unten finden sich aber ein paar Beispiele, wie man die Type Library in ausgewählte Umgebungen einbindet.

Auslieferung mit Sage 50

Bis und mit Sage 50 V2012 wurde die FibuSDK Type Library als eigenständiges File (FibuSDK.tlb) augeliefert. Ab Sage 50 V2013 ist die Type Library in die FibuSDK COM DLL (fbsdk50.dll) eingebettet und wird nicht mehr separat ausgeliefert.

Dies hat den Vorteil, dass die Type Library automatisch mit-registriert wird, wenn die COM DLL mit regsvr32.exe registriert wird. Da das Sage 50 Setup die COM DLL selbständig registriert, ist die Type Library somit nach einer Installation von Sage 50 ebenfalls bereits registriert.

Type Library manuell registrieren

Will man die Type Library aus irgendeinem Grund manuell registrieren (oder de-registrieren), kann man folgendermassen vorgehen:

1. Eine Command Shell (cmd.exe) mit Administrator-Rechten starten
2. Folgendes Kommando ausführen, um die Type Library zu registrieren (bei Bedarf den Pfad zur DLL anpassen):
   regsvr32.exe "C:\Program Files\Sage\Sage50\Prog\fbsdk50.dll"
3. Um die Type Library zu de-registrieren, gibt man beim obigen Kommando noch den Schalter /u an:
   regsvr32.exe /u "C:\Program Files\Sage\Sage50\Prog\fbsdk50.dll"

Einbinden der Type Library

C++ / Visual C++

Im stark typisierten C++ ist es zwingend notwendig, die Type Library einzubinden. In MS Visual C++ tut man dies mit folgendem Statement:

```
#import <fbsdk50.dll>
```

Damit die DLL gefunden wird, muss im C++ Projekt zusätzlich ein Include-Pfad auf das Verzeichnis gesetzt werden, wo die DLL liegt (üblicherweise C:\Program Files (x86)\Sage\Sage50\Prog).

.NET (C#, Visual Basic .NET)

In .NET Programmiersprachen (z.B. C# oder Visual Basic .NET) ist das Einbinden der Type Library nicht zwingend notwendig, macht aber die Programmierung wesentlich einfacher. Um die Type Library in ein Projekt einzubinden muss man eine Referenz darauf erzeugen. Eine Referenz wird wie folgt erzeugt (die einzelnen Schritte sind auf MS Visual Studio 2010 abgestimmt, können sinngemäss aber auch auf andere Versionen von Visual Studio übertragen werden):

- In Visual Studio einen Rechtsklick auf das .NET Projekt machen
- Aus dem Kontextmenü „Add Reference…“ auswählen (auf Deutsch: „Verweis hinzufügen…“)
- Man hat jetzt 2 Möglichkeiten, bei beiden ist das Resultat das gleiche:
  - Den Reiter „COM“ auswählen. In der Liste den Eintrag „SAGE 50 Object Library“ suchen und „OK“ klicken
  - Den Reiter „Browse“ auswählen (auf Deutsch: „Durchsuchen“). Anschliessend zur COM DLL navigieren (z.B. C:\Program Files (x86)\Sage\Sage50\Prog\fbsdk50.dll), diese auswählen und „OK“ klicken
- Dadurch erhält man in den Referenzen einen Eintrag „fibusdk“. Dieser Eintrag verweist auf die Datei Interop.fibusdk.dll, welche innerhalb des Projekts generiert wurde und in der die Informationen aus der FibuSDK Type Library enthalten sind.

Wichtiger Hinweis: Installiert man später eine neuere Version von Sage 50, so muss man die Referenz erneuern, um Zugriff zu erhalten auf Objekte, Methoden, Properties etc. die in der neuen Version von Sage 50 in FibuSDK neu hinzugekommen sind.

VBA/Visual Basic for *Applications* (z.B. in Excel)

Für VBA gilt das gleiche wie für .NET: Das Einbinden der Type Library ist zwar nicht zwingend notwendig, erleichtert das Programmieren aber ungemein. Die Type Library wird hier wie folgt eingebunden (am Beispiel von Excel):

- Excel starten und das .xls Dokument öffnen, das den VBA Code enthält
- Den VBA Editor öffnen (Alt+F11)
- Im Menü „Extras“ den Eintrag „Verweise…“ auswählen
- Man hat jetzt 2 Möglichkeiten, die beide das gleiche Resultat zur Folge haben:
  - In der Liste der verfügbaren Verweise den Eintrag „SAGE 50 Object Library“ suchen, den Eintrag aktivieren und „OK“ klicken
  - Auf „Durchsuchen“ klicken. Anschliessend zur COM DLL navigieren (C:\Program Files (x86)\Sage\Sage50\Prog\fbsdk50.dll), diese auswählen und „OK“ klicken. Der Eintrag „SAGE 50 Object Library“ wird jetzt in der Liste automatisch ausgewählt und aktiviert.

1.6 History

1.6.1 Neue Features in Version 2019

Keine

1.6.2 Neue Features in Version 2018

Neue Lese-Operationen PREVIOUS/NEXT (LesenRel(10) und LesenRel(11)) zum Lesen des vorherigen/nächsten Datensatzes (erst ab V2018 Auto Update 7). Diese Lese-Operationen bieten eine wesentlich höhere Performance beim Verwenden des In-Memory Cache (nur SQL Mandanten) als die bisherigen Varianten LESS/GREATER (LesenRel(2) und LesenRel(3)). **VORSICHT:** Man darf nicht einfach überall LESS/GREATER durch PREVIOUS/NEXT ersetzen, da es Unterschiede in der Anwendung der Lese-Operationen gibt! Bitte lesen Sie zuerst den neuen Abschnitt „1.8 Die Funktion LesenRel“ durch: Der Abschnitt enthält wesentliche Informationen zum Verständnis, wie Daten gelesen werden, insbesondere wird der Unterschied zwischen LESS/GREATER und PREVIOUS/NEXT erklärt.

Abschnitt „1.2.2 Fehlerquellen“ überarbeitet (ab V2018 Auto Update 7). Den Abschnitt „Benötigte DLLs“ entfernt, da die Informationen darin veraltet waren. Den Abschnitt zur Windows Registry für moderne Windows-Systeme aktualisiert.

Neue Eigenschaften beim Objekt OP (erst ab V2018 Auto Update 2)

- Zahlungsgrund_1
- Zahlungsgrund_2
- Zahlungsgrund_3
- Zahlungsgrund_4
- ReferenzEndToEnd
- ReferenzEndToEndOhneFormat

Objekt Mandant: In-Memory Cache ein- und ausschalten (nur SQL Mandanten)

1.6.3 Neue Features in Version 2017

Keine

1.6.4 Neue Features in Version 2015

Geänderte Methoden

Objekt Beleg, Methode BuchenKst2: Der Parameter KstID2 darf bis auf weiteres nicht mehr verwendet werden. Dies ist auf einen Fehler zurückzuführen, der mit der Einführung

von Verteilmethoden entstanden ist. Der Fehler wird in einer zukünftigen Version von FibuSDK korrigiert werden, ab dann kann KstID2 voraussichtlich wieder verwendet werden.

1.6.5 Neue Features in Version 2014

Keine

1.6.6 Neue Features in Version 2013

Type Library wird nicht mehr als eigenständiges File ausgeliefert

Bis und mit V2012 wurde die FibuSDK Type Library als eigenständiges File (FibuSDK.tlb) augeliefert. Ab V2013 ist die Type Library in die FibuSDK COM DLL (fbsdk50.dll) eingebettet und wird nicht mehr separat ausgeliefert.

Auf laufende Programme, die mit der FibuSDK erstellt wurden, hat diese Änderung keine Auswirkungen. Will man aber eine Änderung am Programmcode vornehmen und anschliessend das Programm neu übersetzen, so muss man unter Umständen die Type Library neu einbinden. *Grundsätzlich gilt, dass Verweise auf das alte Type Library File FibuSDK.tlb geändert werden müssen in Verweise auf die COM DLL fbsdk50.dll.*

Im Kapitel „Die Type Library“ weiter oben findet man einige konkrete Beispiele, wie man einen Verweis auf die Type Library in den Programmiersprachen C++ (Visual C++), C#, VB.NET oder VBA setzt. Es wird auch erklärt, wie man die Type Library im System registrieren kann, falls dies aus irgendeinem Grund nötig sein sollte.

Neue Objekte

Keine

Neue Eigenschaften

Objekt Beleg: LastUserChanged, Vorerfassen

Objekt Buchung: LastUserChanged

Bank: KreditLimite

Geänderte Eigenschaften

Objekt Kreis: BelegNr, KreisID und KontoNr sind neu „nur Lesen“

Objekt Buchung: IstHaben, KstIndex, SteuerIndex und Typ sind neu „Lesen und schreiben“. IstFwBuchung ist neu „nur Lesen“.

Objekt EBanking: Datenpfad, Host und PoolPfad sind neu „nur Lesen"

Objekt Steuerformel: FormelID ist neu „nur Lesen“

Objekt Bank: IBAN ist neu 35 statt 22 Zeichen lang.

Objekt BankInfo: IBAN ist neu 35 statt 22 Zeichen lang. BVId ist neu 13 statt 5 Zeichen lang

Neue Methoden

Objekt Mandant: BelegVerbuchen

Objekt Beleg: SetBookingBlockedCheck

Objekt Buchung: GetNoFwKonto

Geänderte Methoden

Objekt BankInfo: Lesen: Hat neuen Parameter PkNr, BVId Parameter ist neu 13 statt 5 Zeichen lang

1.6.7 Neue Features in Version 2012

Keine

1.6.8 Neue Features in Version 2011

Neue Eigenschaften

Objekt OP: MahnverfahrenID

Objekt PK: MahnverfahrenID

1.6.9 Neue Features in Version 8.0 (2008)

Neue Objekte

Kontakt, Anschrift, PlzStamm, Dauerauftrag

Neue Eigenschaften

Objekt OP: Mitarbeiter, Kontakt

Objekt Beleg: Stornieren

Diverses

Objekt Adresse: komplett überarbeitet

Objekt Buchung: Beträge schreibbar

1.6.10 Neue Features in Version 7.0 (2006)

Neue Objekte

BankInfo, ESRParser, IBANParser, EBanking

Neue Eigenschaften

Objekt Mandant: PfadVorjahr, PfadFolgejahr

Objekt PK: EbppId, Rechnungsart, Mahngebuehrlock, IstEinmalKunde, Bankverbindung

Objekt OP: IBAN, RechnungsArt

Objekt Bankstamm: Swift, BankIdHQ

Objekt Budget: BudgetListe

1.6.11 Neue Features in Version 6.0

Neue Objekte

Budget, BuchArchiv

Neue Eigenschaften

Objekt Buchung: DocID, GFNr, TimeStamp

Objekt PK: VkontoNr

Objekt OP: GFNr, OpTyp

Objekt Steuer: SaldoSatz

Objekt Konto: FlagReadOnly

Neue Methoden

Objekt OP: ShowDocument

Erweiterte Methoden

Buchung.SetIndex

1.7 Beispielprogramme

Auf der SOK Diskette finden Sie einige Beispiel Programme, die mit SOK arbeiten.

1.7.1 Auftrag

Ein kleines Excel-Programm mit dem Sie Aufträge / Rechnungen verbuchen können.

Sourcen: \Samples\Excel

1.7.2 FibuOLE

Ein kleines VB-Programm, das die Verwendung von OLE Technologie aufzeigt und den Zugriff auf Sage50-Daten und die Auswertung mittels Excel darstellt.

Sourcen: \Samples\FibuOLE

1.7.3 Minibu

Ein kleines VB-Programm mit dem Sie Belege verbuchen können.

Sourcen: \Samples\MiniBu

1.7.4 OleDemo

Eine Beispielanwendung in C++. Nach dem Start wählen Sie einen Debi- Mandanten in der Datei-Öffnen-Dialogbox. Sofern der Mandant erfolgreich geöffnet wurde, gelangen Sie in einen Dialog.

Sie müssen das Debitorensammelkonto angeben (typischerweise 1050) und eine vorhandene Adresse. In beiden Fällen können Sie mit dem Browse-Button aus den vorhandenen Konten resp. Adressen auswählen. Als letztes geben Sie noch die PK-Bezeichnung ein. Mit Betätigen des Buttons Einfügen wird der PK erstellt. Sie können nachher mit Browse PK nachsehen, ob der PK nun tatsächlich vorhanden ist.

Zu Beachten: Dieses Projekt soll nur zeigen, wie Sie mit Visual C++ das SOK bedienen können. Die Beispielanwendung hat fast keinerlei Fehlerüberprüfung. Es werden auch nur die allerwichtigsten PK-Daten gesetzt.

Die Dateien **Fibusdk.h** und **Fibusdk.cpp** wurden vom Class Wizard automatisch aus der FibuSDK Type Library generiert. Der eigentliche Beispiel-Code befindet sich in der Datei **oleDemoDlg.cpp**. Die Stellen sind im Code mit dreizeiligen Kommentaren markiert.

Sourcen: \Samples\OleDemo

1.8 Die Funktion LesenRel

1.8.1 Typen von Lese-Operationen

Die Mehrzahl der FibuSDK Objekte hat eine Funktion namens LesenRel(), mit der ein Datensatz gelesen werden kann. Die Funktion hat einen Parameter, mit dem der FibuSDK-Benutzer bestimmen kann, welche Art von Lese-Operation verwendet werden soll. Beispiel: GREATER liest den nächst-grösseren Datensatz *gemäss dem aktuellen Sortierschlüssel* (siehe nächsten Abschnitt) Nicht alle FibuSDK Objekte unterstützen alle Lese-Operationen, massgebend ist die LesenRel() Dokumentation des jeweiligen FibuSDK Objekts.

Einige der Lese-Operationen sind **absolut**, d.h. es spielt keine Rolle, was zuvor gelesen wurde. Beispiel: FIRST liest immer den ersten Datensatz gemäss dem aktuellen Sortierschlüssel.

Die meisten der Lese-Operationen sind **relativ**, von daher kommt auch der Funktionsname Lesen**Rel**(). Einige Lese-Operation sind relativ zur Lese-Position, die durch eine frühere Lese-Operation etabliert wurde (z.B. NEXT). Andere Lese-Operationen sind relativ zum Inhalt des Objekts in dem Moment, wo gelesen wird (z.B. GREATER).

1.8.2 Lese-Richtung / Sortierschlüssel / Eindeutigkeit

Die Lese-Richtung von Lese-Operationen wird durch den momentan auf dem FibuSDK Objekt eingestellten Sortierschlüssel bestimmt. Bei vielen Objekten kann der FibuSDk-Benutzer den Sortierschlüssel nicht explizit einstellen, sondern er ist fix eingestellt. Beispiel: Beim Adresse Objekt ist immer die Adress ID als Sortierschlüssel eingestellt.

Bei einigen Objekten kann der FibuSDK-Benutzer den Sortierschlüssel dagegen explizit einstellen. Beispiel: Beim OP Objekt kann der FibuSDK-Benutzer den Sortierschlüssel mit der Funktion SetIndex() einstellen. Der Default-Sortierschlüssel ist die OP-Nummer, aber der FibuSDK-Benutzer kann auch die OP-Referenz als alternativen Sortierschlüssel einstellen.

Die meisten Sortierschlüssel enthalten Werte, die einen Datensatz eindeutig identifizieren. Beispiel: Der Default-Sortierschlüssel beim OP Objekt ist die OP-Nummer, welche einen Offenen Posten eindeutig identifiziert. Wählt der FibuSDK-Benutzer dagegen die OP-Referenz als Sortierschlüssel, so können Duplikate vorhanden sein, d.h. mehrere Offene Posten können die gleiche OP-Referenz haben.

Die meisten Sortierschlüssel bestehen nur aus einem einzigen Merkmal. Beispiel: Beim PK Objekt besteht der Default-Sortierschlüssel aus der PK-Nummer. Einige Sortierschlüssel bestehen dagegen aus mehreren Merkmalen. Beispiel: Beim Buchung Objekt besteht der Default-Sortierschlüssel aus den zwei Merkmalen Beleg-Nummer und Beleg-Index.

1.8.3 Absolute Lese-Operationen

Die folgenden Lese-Operationen sind absolut, d.h. man kann sie mehrmals hintereinander aufrufen und erhält als Resultat immer den gleichen Datensatz:

- FIRST: Liest den ersten Datensatz gemäss dem aktuellen Sortierschlüssel. Erlaubt der Sortierschlüssel Duplikate, so liest die Operation den Datensatz mit

dem ersten Duplikat – mit grosser Wahrscheinlichkeit ist das derjenige Datensatz, unter den Duplikaten, der zuerst eingefügt worden ist.

- LAST: Liest den letzten Datensatz gemäss dem aktuellen Sortierschlüssel. Erlaubt der Sortierschlüssel Duplikate, so liest die Operation den Datensatz mit dem letzten Duplikat – mit grosser Wahrscheinlichkeit ist das derjenige Datensatz, unter den Duplikaten, der zuletzt eingefügt worden ist.

1.8.4 Lese-Operationen relativ zur Lese-Position

Die folgenden Lese-Operationen sind relativ zur Lese-Position, die von einer vorherigen Lese-Operation etabliert worden ist. Die folgenden Lese-Operationen funktionieren also nur dann, wenn mit dem FibuSDK Objekt zuvor mindestens einmal eine andere Lese-Operation ausgeführt worden ist. Der Inhalt des FibuSDK Objekts in dem Moment, wo gelesen wird, spielt keine Rolle.

- PREVIOUS: Liest den vorherigen Datensatz gemäss dem aktuellen Sortierschlüssel. Erlaubt der Sortierschlüssel Duplikate, so liest die Operation den vorherigen Datensatz innerhalb der Duplikate mit dem gleichen Schlüsselwert. Mit anderen Worten: Auch bei Duplikaten überspringt die Operation keine Datensätze, sondern erfasst immer alle Datensätze (vgl. die relative Lese-Operation LESS). Beispiel: Um alle Datensätze in umgekehrter Reihenfolge des Sortierschlüssels zu lesen kann er FibuSDK Benutzer zuerst ein einzelnes LesenRel(LAST) ausführen, danach gefolgt von einer while-Schlaufe, in der bei jedem Durchlauf LesenRel(PREVIOUS) ausgeführt wird.

- NEXT: Liest den nächsten Datensatz gemäss dem aktuellen Sortierschlüssel. Erlaubt der Sortierschlüssel Duplikate, so liest die Operation den nächsten Datensatz innerhalb der Duplikate mit dem gleichen Schlüsselwert. Mit anderen Worten: Auch bei Duplikaten überspringt die Operation keine Datensätze, sondern erfasst immer alle Datensätze (vgl. die relative Lese-Operation GREATER). Beispiel: Um alle Datensätze in der Reihenfolge des Sortierschlüssels zu lesen kann er FibuSDK Benutzer zuerst ein einzelnes LesenRel(FIRST) ausführen, danach gefolgt von einer while-Schlaufe, in der bei jedem Durchlauf LesenRel(NEXT) ausgeführt wird.

Hinweis: Diese Lese-Operationen sind erst ab Sage 50 V2018 Auto Update 7 verfügbar.

1.8.5 Lese-Operationen relativ zum Objekt-Inhalt

Die folgenden Lese-Operationen sind relativ zum Inhalt des FibuSDK Objekts in dem Moment, wo gelesen wird. Der FibuSDK Benutzer kann den Inhalt des FibuSDK Objekts entweder manuell abfüllen, oder er kann zuerst eine andere Lese-Operation ausführen, welche das FibuSDK Objekt mit Inhalt füllt. Damit die folgenden Lese-Operationen funktionieren, muss man mit dem FibuSDK Objekt zuvor also **keine** andere Lese-Operation ausgeführt worden ist.

- EQUAL: Liest denjenigen Datensatz, dessen Merkmale gemäss dem aktuellen Sortierschlüssel übereinstimmen mit dem Inhalt der entsprechenden Merkmale des FibuSDK Objekts. Erlaubt der Sortierschlüssel Duplikate, so liest die Operation den Datensatz mit dem ersten Duplikat – mit grosser Wahrscheinlichkeit ist das derjenige Datensatz, unter den Duplikaten, der zuerst eingefügt worden ist.

- LESS: Liest denjenigen Datensatz, dessen Merkmale gemäss dem aktuellen Sortierschlüssel «kleiner» sind als der Inhalt der entsprechenden Merkmale des FibuSDK Objekts. Erlaubt der Sortierschlüssel Duplikate, so überspringt die Operation alle Duplikate und liest den letzten Datensatz, dessen Merkmale «kleiner» sind. Die Art, wie auf «kleiner» verglichen wird, unterscheidet sich je nach Datentyp der involvierten Merkmale. Beispiele: Bei einem numerischen Datentyp findet ein numerischer «kleiner» Vergleich statt, bei einem String-Datentyp findet ein lexikalischer «kleiner» Vergleich statt.

- GREATER: Liest denjenigen Datensatz, dessen Merkmale gemäss dem aktuellen Sortierschlüssel «grösser» sind als der Inhalt der entsprechenden Merkmale des FibuSDK Objekts. Erlaubt der Sortierschlüssel Duplikate, so überspringt die Operation alle Duplikate und liest den ersten Datensatz, dessen Merkmale «grösser» sind. Die Art, wie auf «grösser» verglichen wird, unterscheidet sich je nach Datentyp der involvierten Merkmale. Beispiele: Bei einem numerischen Datentyp findet ein numerischer «grösser» Vergleich statt, bei einem String-Datentyp findet ein lexikalischer «grösser» Vergleich statt.

- LESSOREQUAL: Verhält sich gleich wie LESS, ausser die Lese-Operation auf einem «kleiner oder gleich» statt einem «kleiner» Vergleich basiert.

- GREATEROREQUAL: Verhält sich gleich wie GREATER, ausser die Lese-Operation auf einem «grösser oder gleich» statt einem «grösser» Vergleich basiert.

1.8.6 Unterschiede zwischen LESS/GREATER und PREVIOUS/NEXT

Wie unterscheiden sich die Lese-Operationen LESS/GREATER von den Lese-Operationen PREVIOUS/NEXT?

- Sowohl LESS/GREATER als auch PREVIOUS/NEXT lesen gemäss dem aktuellen Sortierschlüssel. LESS/GREATER verwenden aber als Basis den Inhalt des FibuSDK Objekts in dem Moment, wo gelesen wird, während PREVIOUS/NEXT als Basis den Lese-Zeiger der vorhergehenden Lese-Operation verwenden.

- LESS/GREATER brauchen keine vorhergehende Lese-Operation. PREVIOUS/NEXT brauchen zwingend eine vorhergehende Lese-Operation, um den Lese-Zeiger zu etablieren.

- Bei einem Sortierschlüssel, der Duplikate erlaubt, überspringen LESS/GREATER Datensätze mit Duplikat-Werten. PREVIOUS/NEXT lesen dagegen immer den tatsächlichen gemäss Sortierschlüssel vorhergehenden/nächsten Datensatz. Bei einem Sortierschlüssel, der Duplikate erlaubt, ist es mit LESS/GREATER also nicht möglich, alle Datensätze z.B. in einem Loop auszulesen.

- PREVIOUS/NEXT sind erst ab Sage 50 V2018 Auto Update 7 verfügbar. LESS/GREATER waren bereits in früheren Sage 50 Versionen verfügbar, waren aber irreführend als PREV/NEXT dokumentiert.

- Bei eingeschaltetem In-Memory Cache (siehe Kapitel 1.9) sind LESS/GREATER zur Zeit langsamer als mit ausgeschaltetem In-Memory-Cache. PREVIOUS/NEXT erzielen dagegen bei eingeschaltetem In-Memory Cache den erwarteten

Performance-Gewinn. Es ist geplant, diese Einschränkung von LESS/GREATER in einer zukünftigen Version von Sage 50 zu eliminieren.

1.8.7 Sortierschlüssel mit Duplikaten vs. LESS/GREATER

Am folgenden Beispiel wird erläutert, warum die Lese-Operationen LESS/GREATER bei Sortierschlüsseln, die Duplikate erlauben, unter Umständen ein für den FibuSDK-Benutzer unerwartetes Ergebnis liefern.

In der Buchungs-Tabelle kann man Datensätze sortiert nach der Dokumenten ID lesen, wenn man zuvor auf dem Buchung Objekt SetIndex(5) ausgeführt hat. Die Dokumenten ID ist aber für alle Buchungen innerhalb eines Belegs gleich, d.h. der Sortierschlüssel lässt Duplikate zu. Ein Ausschnitt aus der Buchungs-Tabelle sieht z.B. so aus:

Doc ID	Record ID	Beleg-Nummer	Beleg-Index
111	17	14	0
111	18	14	1
222	76	30000	0
222	77	30000	1
222	78	30000	2
333	4	2	0
333	5	2	1

Hat man mit dem Buchung Objekt zuvor den Datensatz mit Record ID 76 gelesen und führt jetzt die Lese-Operation GREATER aus, dann erhält man nicht den Datensatz mit Record ID 77, wie man das vielleicht erwarten könnte, sondern den Datensatz mit Record ID 4. Die zwei dazwischenliegenden Datensätze mit den Record IDs 77 und 78 werden übersprungen. Der Grund ist, dass beim Lesen von Datensatz mit Record ID 76 das Merkmal DocId mit dem Wert „222“ gefüllt wurde, und GREATER als nächst grössere Doc Id somit „333“ ermittelt.

Die folgenden Objekte erlauben den Einsatz von Sortierschlüsseln mit Duplikaten:

- Objekt Buchung
    - Mit der Funktion SetIndex(5) wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der Dokumenten ID.
- Objekt BuchArchiv
    - Mit dem Setter des Properties "TimeStamp" wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach dem Datum, an dem die Buchung archiviert wurde.

- Mit dem Setter des Properties "BelegNr" wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der original Beleg-Nr., d.h. der Beleg-Nummer, die die Buchung hatte, bevor sie archiviert wurde.

- Objekt Budget

  - Ein neues OBudget Objekt hat per Default bereits einen Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der Konto ID. Da man ein Konto über mehrere Kostenstellen budgetieren kann ist das Sortieren nur nach Konto ID nicht eindeutig.

- Objekt OP

  - Mit der Funktion SetIndex() wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der Referenz-Nr.

- Objekt PK

  - Mit der Funktion SetIndex(1) wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der PK-Index-Nr.

  - Mit der Funktion SetIndex(2) wird ein Sortierschlüssel gesetzt, der Duplikate erlaubt. Der Sortierschlüssel erlaubt das Lesen nach der Adresse ID.

1.9 In-Memory Cache für SQL Mandanten

1.9.1 Was ist der In-Memory Cache?

Die FibuSDK Schnittstelle ist historisch bedingt sehr stark auf das sogenannte «ISAM» Datenzugriffsmodell zugeschnitten, bei dem viele kleine Lese- und Schreib-Operationen mit individuellen Datensätzen gemacht werden. Ein FibuSDK Anwender, der z.B. die Daten der ersten 100 Kontos auslesen will, muss dazu selber eine Schleife programmieren, die solange Lese-Operationen ausführt, bis die Abbruchbedingung (100 Kontos) erreicht ist. Es resultieren 100 Lese-Operationen.

Die Pervasive Btrieve Datenbank-Engine (die bei Btrieve Mandanten zum Einsatz kommt) ist sehr performant wenn es darum geht, solche kleinen Lese- und Schreib-Operationen durchzuführen. Der Grund ist, dass die Pervasive Btrieve Datenbank-Engine ebenfalls ISAM-basiert ist. Die FibuSDK-Schnittstelle versteht sich sozusagen «sehr gut» mit Pervasive Btrieve.

Moderne relationale Datenbank-Systeme wie z.B. Microsoft SQL Server (der bei SQL Mandanten zum Einsatz kommt) sind im Gegensatz dazu aber besonders performant, wenn man viele Daten auf einmal abfragt. Sie sind nicht darauf ausgelegt, einzelne Datensätze sequenziell zu lesen. Im obigen Beispiel zum Lesen von 100 Kontos reagiert ein relationales Datenbank-System wie z.B. SQL Server deshalb sehr träge auf die 100 einzelnen Lese-Operationen und benötigt wesentlich länger für die Verarbeitung als die Pervasive Btrieve Datenbank-Engine.

Der In-Memory Cache ist ein Mechanismus, mit dem der FibuSDK-Anwender in spezifischen Anwendungsfällen dafür sorgen kann, dass bei SQL Mandanten nicht mehr alle Lese-Operationen an das relationale Datenbank-System weitergegeben werden, sondern dass Daten zuerst in einen Cache im Speicher (RAM) geladen und danach von dort gelesen werden. Der Flaschenhals des relationalen Datenbank-Systems wird dadurch zu einem wesentlichen Teil ausgeschaltet, was das lesende Verarbeiten von grossen Datenmengen via FibuSDK signifikant beschleunigt. An die Geschwindigkeit der Pervasive Btrieve Datenbank-Engine kommt aber auch der In-Memory Cache nicht heran.

1.9.2 Wie funktioniert der In-Memory Cache?

Hinter der Fassade der FibuSDK Schnittstelle versteckt agiert das Datenzugriffs-Layer von Sage 50 Rechnungswesen. Im Normalfall muss sich der FibuSDK Anwender nicht um Details kümmern, die das Datenzugriffs-Layer betreffen. Der Ablauf einer Daten-Lese-Operation ist folgender:

1. Die vom FibuSDK Anwender programmierte Applikation ruft eine Lese-Funktion im FibuSDK auf.

2. Das FibuSDK trifft intern die nötigen Vorbereitungen, damit die gewünschten Daten gelesen werden können.

3. Das FibuSDK gibt den Auftrag zum Lesen der Daten an das Datenzugriffs-Layer weiter.

4. Das Datenzugriffs-Layer gibt den Auftrag an das Datenbank-Layer weiter.

    - Btrieve Mandanten: Das Datenbank Layer ist die Pervasive Btrieve Datenbank-Engine.

    - SQL Mandanten: Das Datenbank Layer ist die SQL Server Datenbank-Engine (oder in Zukunft unter Umständen ein anderes relationales Datenbanksystem wie z.B. MySQL)

5. Das Resultat der Lese-Operation wird in umgekehrter Reihenfolge zurückgegeben, bis es wieder bei der Applikation angelangt ist.

Nur für SQL Mandanten kann der FibuSDK Anwender das Datenzugriffs-Layer instruieren, dass ab jetzt der In-Memory Cache eingeschaltet bzw. wieder ausgeschaltet werden soll. Im Normalzustand ist der In-Memory Cache ausgeschaltet.

Während der In-Memory Cache eingeschaltet ist, gibt das Datenzugriffs-Layer im Schritt 4 des oben beschriebenen Ablaufs nicht mehr jede Lese-Operation an das Datenbank-Layer weiter, sondern verhält sich wie folgt:

- **Schreib-Operationen sind nicht erlaubt! Erhält das Datenzugriffs-Layer die Aufforderung zum Schreiben eines Datensatzes, so wirft es eine Exception! Auf welche Art die Exception bei der Applikation ankommt, welche den illegalen Schreibzugriff ausführt, ist hier nicht weiter beschrieben, da COM Exceptions in verschiedenen Programmierumgebungen unterschiedlich gehandhabt werden.**

- Das Datenzugriffs-Layer ermittelt, aus welcher Datenbank-Tabelle die angeforderten Daten gelesen werden sollen.

- Das Datenzugriffs-Layer prüft, ob die Daten der Datenbank-Tabelle bereits im In-Memory Cache vorhanden sind.

    - Falls nein, so lädt das Datenzugriffs-Layer den Inhalt der **gesamten** Datenbank-Tabelle in den In-Memory Cache.

    - Falls ja, so unternimmt das Datenzugriffs-Layer nichts weiter.

- Das Datenzugriffs-Layer liest jetzt die angeforderten Daten aus dem In-Memory Cache.

Der In-Memory Cache wird also nach dem Einschalten nach und nach mit Daten befüllt, so dass mit der Zeit immer weniger Datenbank-Zugriffe notwendig sind und die Daten immer direkt aus dem In-Memory Cache gelesen werden können.

Es werden immer die gesamten Daten einer Datenbank-Tabelle in den In-Memory Cache geladen, auch wenn nur ein einziger Datensatz der Datenbank-Tabelle gelesen werden soll. Die Effizienz des Ladens in den In-Memory Cache wird dadurch maximiert, bei gleichzeitig minimalem Aufwand zur Überprüfung, ob die angefordeten Daten bereits im In-Memory Cache sind. Der Nachteil ist erhöhter Speicherbedarf.

Wird das Datenzugriffs-Layer instruiert, den In-Memory Cache wieder abzuschalten, so verwirft es die Daten im In-Memory Cache vollständig.

1.9.3 Einschränkungen

Wie bereits weiter oben erwähnt kann der In-Memory Cache nur für SQL Mandanten verwendet werden, da auch nur bei diesen ein Performance-Problem vorhanden ist. Wird der In-Memory Cache für Btrieve Mandanten eingeschaltet, so hat das keinen Effekt, das Einschalten wird einfach ignoriert.

Ebenfalls bereits weiter oben erwähnt, aber hier noch einmal besonders erwähnt werden muss, dass während der In-Memory Cache eingeschaltet ist, **keinerlei Schreib-Operationen ausgeführt werden dürfen!** Der Grund ist, dass Daten im In-Memory Cache beim Ausschalten des Cache verworfen und nicht zurück in die Datenbank geschrieben werden. Schreib-Operationen können auch nicht einfach an die Datenbank weitergeleitet werden, da sonst die Daten im In-Memory Cache nicht mehr mit der Datenbank übereinstimmen. Die Daten im In-Memory Cache bei einer Schreib-Operation zu invalidieren und danach erneut aus der Datenbank zu lesen wäre zwar technisch möglich, hätte aber die Cache-Implementation wesentlich verkompliziert und zu Performance-Einbussen geführt. Aus all diesen Gründen hat sich Sage Schweiz gegen die Unterstützung von Schreib-Operationen entschieden.

Die Lese-Operationen LesenRel(LESS) und LesenRel(GREATER) sollten nicht zusammen mit dem In-Memory Cache verwendet werden, da sie bei eingeschaltetem Cache sogar **langsamer** sind als bei ausgeschaltetem Cache! Es ist geplant, diese Einschränkung in einer zukünftigen Version von Sage 50 zu eliminieren, bis dahin sollten Sie allerdings eine der folgenden alternativen Lese-Operationen verwenden:

- LesenRel(PREVIOUS) statt LesenRel(LESS) zum Lesen des vorherigen Datensatzes.

- LesenRel(NEXT) statt LesenRel(GREATER) zum Lesen des nächsten Datensatzes.

1.9.4 Hinweis für die Verwendung

Der FibuSDK Anwender kann den In-Memory Cache nur für Programm-Funktionen verwenden, die sowohl einen **wohl-definierten Anfang** als auch ein **wohl-definiertes Ende** haben.

Zum Anfang der Programm-Funktion wird der In-Memory Cache eingeschaltet.

Zum Ende der Programm-Funktion wird der In-Memory Cache wieder ausgeschaltet (das darf nicht vergessen werden!).

Dazwischen darf die Programm-Funktion keine Schreib-Operationen ausführen.

1.9.5 Wie wird der In-Memory Cache ein- und ausgeschaltet?

Das Mandant Objekt hat eine Eigenschaft namens «InMemoryCacheEnabled». Die Eigenschaft kann bei angemeldetem SQL Mandant auf «true» bzw. «false» gesetzt weden, um den Cache ein- bzw. auszuschalten.

Siehe auch die Beschreibung der Eigenschaft weiter unten in diesem Dokument, dort wo das Mandant Objekt dokumentiert ist.

2.0 SOK Referenz

In diesem Teil der Dokumentation sind alle SOK Objekte in alphabetischer Reihenfolge aufgeführt und im Detail beschrieben. Dieser Referenzteil ist auch als Windows Help-Datei (fibusdk.hlp) auf Ihrem System installiert.

2.1 Adresse

2.1.1 Adresse Objekt

Bezeichnung Mit diesem Objekt können interne oder externe Adressen bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuAdresse erstellt. Die Eigenschaften und Methoden in diesem Objekt, die mit einem Stern (*) in der Beschreibung anfangen, funktionieren nur wenn ein DebiNT-/KrediNT-Mandant mit der internen Adressverwaltung arbeitet. Wenn eine Anwendung auch mit Mandanten arbeiten soll, die eine externe Adressverwaltung verwenden, dürfen keine dieser Funktionen verwendet werden.

Eigenschaften Anrede: String[31] (*)

Die Anrede

Briefanrede: String[65] (*)

Die Briefanrede

Code: String[9] (*)

Der Code

Druckinfo: Integer

(*) Der Adress Druck-Code

Email: String[61]

(*) EMail-Adresse

Firma: String[31] (*)

Die Firma

Homepage: String[61]

(*) Internetadresse der Homepage.

ID: String[13]

(*) Der eindeutige Kürzel der Adresse.

IstIntern: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn für den aktuellen Mandant die interne Adressverwaltung aktiv ist. In diesem Fall können die, mit Stern (*) gekennzeichneten Eigenschaften und Methoden verwendet werden. Diese Eigenschaft verändert sich nur, wenn ein neuer Mandant geöffnet wird. (siehe Mandant.Login)

Land: String[3]

(*) Der Ländercode der vor die Adresse.PLZ eingefügt wird. z.B CH oder D

MahnSperrCode: Boolean

(*) Das Mahnsperrcode-Flag

PLZ: String[7]

(*) Die Postleitzahl

Ort: String[25]

(*) Die Ortschaft

SortID: String[11]

(*) Der Sortier-Schlüssel. Eine kopie dieses Feldes befindet sich in der PK-Datei von DebiNT oder KrediNT.

Sprache: String[1]

(*) Der Sprach-Code

StampUser: String[1]

(*) Der Benutzer der diesen Datensatz zuletzt gespeichert hat.

StampDate: Date

(*) Das Datum an dem dieser Datensatz zuletzt gespeichert wurde.

Strasse: String[31] (*)

Die Strasse

Telefon1: String[21]

(*) Die Telefonnummer: Geschäft

Telefon2: String[21]

(*) Die Telefonnummer: Privat

Telefon3: String[21]

(*) Die Telefonnummer: Natel

Telefon4: String[21]

(*) Die Telefonnummer: Telefax

Zusatz1: String[31]

(*) Der erste Zusatz für die Postadresse.

Zusatz2: String[31]

(*) Der zweite Zusatz für die Postadresse.

2.1.2 Adresse.Browser

Integer Browser *(AdressID)*

Mit dieser Methode kann ein Benutzer mit Hilfe eines Browsers bequem eine Adresse suchen und lesen.

Rückgabe 0, wenn eine Adresse gelesen wurde, sonst ein Fehlerstatus.

Parameter *AdressID*: String[13]

Enthält die aktuelle Adresse des Browsers. Ist dieser Parameter ein leerer String, steht der Browser auf dem ersten Datensatz.

Siehe Lesen, LesenRel

2.1.3 Adresse.Einfuegen

Integer Einfuegen()

(*) Schreibt eine neue Adresse in die Datei. Der ID dieses Objekts darf noch nicht von einer anderen Adresse verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Adresse, Schreiben

2.1.4 Adresse.Info

String Info *(Code)*

Mit dieser Funktion werden sämtliche Informationen der Adresse zurück gegeben. Bevor diese Methode verwendet wird, muss die Adresse mit der Methode Lesen oder LesenRel gelesen werden.

Rückgabe Die gewünschte Information, ansonsten ein leerer String.

Parameter *Code*: Integer

Die gewünschte Information. Folgende Codes sind definiert:

0 - Der Primäre Adress-Schlüssel

1 - Firma

2 - Anrede

3 - Zusatz1

4 - Zusatz2

5 - Stasse

6 - Plz Ort

7 - Volle Briefanrede

8 - Telefon-Geschäft

9 - Telefon-Privat

10 - Natel

11 - Telfax

20 - Sprach-Code

21 - Ein Länderpräfix für Code 6, z.B. 'CH'

99 - Der Adress Sortier-Schlüssel

100 - Adresse gem. Druckinfo

101 - 1 zeilige Adresse

102 - 2 zeilige Adresse

103 - 3 zeilige Adresse

106 - Die Post Adresse

Kommentar Es ist möglich, dass eine Fehlermeldung als Info zurück gegeben wird. Diese enthält zwei Sonderzeichen: "$#Adresse nicht gelesen."

Beispiel Eine Adresse lesen, und den Firmennamen holen:

```
Dim Adresse As Object
Set Adresse = Mandant.AdresseNeu()
res% = Adresse.Lesen(70001)
Firma$ = Adresse.Info(1)
```

Siehe Adresse

2.1.5 Adresse.Lesen

Integer Lesen *(AdressID)*

Mit dieser Funktion können Sie eine bestimmte Adresse lesen. Danach können die einzelnen Informationen mit Info geholt werden.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *AdressID*: String[13]

Die Adress-Nr oder ID.

Siehe Adresse, LesenRel, Browser

2.1.6 Adresse.LesenRel

Integer LesenRel *(Wie)*

(*) Liest eine Adresse von der Datei relativ zu der Adresse, die sich jetzt schon in diesem Adresse-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Adresse gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

4 (LAST): Liest den letzten Datensatz.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 und Wie = 4 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Adresse, Lesen, Browser

2.1.7 Adresse.Loeschen

Integer Loeschen()

(*) Löscht den aktuellen Datensatz in diesem Objekt. Die Adresse muss zuerst gelesen werden bevor sie gelöscht werden kann.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Kommentar Diese Funktion kontrolliert nicht, ob eine Adresse schon von einem PK verwendet wurde. Wenn das der Fall wäre, sollten Sie die Adresse nicht löschen.

2.1.8 Adresse.Schreiben

Integer Schreiben()

(*) Schreibt die zuvor gelesene Adresse in die Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Adresse, Einfuegen

2.2 Anschrift

2.2.1 Anschrift Objekt

Bezeichnung Mit diesem Objekt können Anschriften von Kontakten bearbeitet werden. Objekt wird mit der Funktion Mandant.NeuObjekt(25) erstellt

Eigenschaften AnschriftNr: Integer

Die Anschrift-Nr wird benützt, um eine Anschrift eindeutig zu identifizieren.

AnschriftTyp: Integer

Folgende Anschrifttypen sind definiert:

1 = Geschäft

2 = Privat

Briefanrede: String[65]

Die Briefanrede

Email: String[61]

EMail-Adresse

KontaktNr: String[18]

Der Kontakt zu der diese Anschrift gehört.

Land: String[3]

Der Ländercode der vor die Adresse.PLZ eingefügt wird. z.B CH oder D

PLZ: String[7]

Die Postleitzahl

Ort: String[25]

Die Ortschaft

Sprache: String[1]

Der Sprach-Code

Strasse: String[31]

Die Strasse

Telefon1: String[21]

Die Telefonnummer: Geschäft

Telefon2: String[21]

Die Telefonnummer: Privat

Telefon3: String[21]

Die Telefonnummer: Natel

Telefon4: String[21]

Die Telefonnummer: Telefax

URL: String[61]

URL-Adresse der Hompage

Zusatz1: String[31]

Der erste Zusatz für die Postadresse.

Zusatz2: String[31]

Der zweite Zusatz für die Postadresse.

2.2.2 Anschrift.Einfuegen

Integer Einfuegen()

Schreibt eine neue Anschrift in die Datei. Die AnschriftNr dieses Objekts darf noch nicht von einem anderen Eintrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Anschrift, Schreiben

2.2.3 Anschrift.Lesen

Integer Lesen *(AnschriftNr)*

Liest einen Anschrift von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *AnschriftNr*: Integer

Die Anschrift-Nr der Anschrift die gelesen werden soll.

Siehe Anschrift, AnschriftNr

2.2.4 Anschrift.LesenRel

Integer LesenRel *(Wie)*

Liest eine Anschrift von der Datei relativ zum Eintrag, der sich jetzt im Anschrift-Objekt befindet. Wenn eine (Kontakt.KontaktNr) festgelegt wird, bevor diese Funktion aufgerufen wird, werden nur Anschriften vom dem angegebenen Kontakt zurückgegeben.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Anschrift gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei, resp des Kontakts.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Beispiel Das C#-Beispiel fügt zwei Anschriften an den Kontakt "2" an und liest anschliessend alle Anschriften von dem Kontakt und zeigt sie in einer Dialog Box an.

```
// Demo-Mandant öffnen
OMandant man = new OMandant ();
man.Login (2, @"C:\ProgramData\Sage\Data\Rewe\SageDemo16", null);

// Anschrift-Objekt holen
Anschrift ans = (Anschrift)man.NeuObjekt (25);

// Erste Anschrift einfuegen
ans.KontaktNr = "10000.000";
ans.AnschriftNr = ans.LetzteAnschriftNr() + 1;

ans.AnschriftTyp = 1;
ans.Telefon1 = "11 111 1111";
ans.EMail = "meyer@business.ch";
short res = ans.Einfuegen ();

// noch eine Anschrift
ans.KontaktNr = "10000.001";
```

```
        ans.AnschriftNr = ans.LetzteAnschriftNr () + 1;
        ans.AnschriftTyp = 2;
    ans.Telefon1 = "22 222 2222";
    ans.EMail = "meyer@privat.ch";
    res = ans.Einfuegen ();

        // jetzt alle Anschriften lesen
    ans.KontaktNr = 2;
        res = ans.LesenRel (0);
        while (res == 0) {
        MessageBox.Show (ans.Telefon1 + ": " + ans.EMail);
        res = ans.LesenRel (3);
        }

        // Sicherstellen, dass alle Ojekte vor
        // Mandant-Objekt verschwinden.
    ans = null;
        GC.Collect ();
    GC.WaitForPendingFinalizers ();
        man.Logout ();
```

Siehe Anschrift, Lesen

2.2.5 Anschrift.LetzteAnschriftNr

Integer LetzteAnschriftNr()

Gibt die zuletzt verwendete AnschriftNr zurück

Rückgabe Die zuletzt verwendete AnschriftNr oder 0, wenn die Anschrift-Tabelle leer ist.

Siehe Anschrift

2.2.6 Anschrift.Loeschen

Integer Loeschen()

Löscht die zuvor gelesene Anschrift aus der Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Anschrift, Einfuegen

2.2.7 Anschrift.Schreiben

Integer Schreiben()

Schreibt die zuvor gelesene Anschrift, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Anschrift, Einfuegen

2.2.8 Anschrift.SetIndex

Integer SetIndex*(Code)*

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code*: Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach AnschriftNr.

1 : Sortiert nach KontaktNr & AnschriftTyp.

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

2.3 Bank

2.3.1 Bank Objekt

Bezeichnung Mit diesem Objekt können die internen Banken von DebiNT und KrediNT bearbeitet werden. Die interne Banken sind eigentlich die Bankkonten mit denen ein OP elektronisch bezahlt werden soll. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(13) erstellt.

Eigenschaften BankId: String[13]

Der ID einer Bank aus dem **Bankstamm**.

BankKonto: String[25]

Die Konto-Nr des Bankkontos.

BankNr: String[5] (Index)

Der eindeutige Kürzel der Bankeintrag in diesem Objekt identifiziert.

IBAN: String[35]

Der IBAN-Account der internen Bank.

KontoNr: String[13]

Das Fibu-Konto in dem Buchungen für diese Bank gespeichert werden.

KreditLimite: Currency

Typ: Interger

Der Typ des Eintrags: 0 - Fibu Konto, 1 - Bank, 2 - Post Check

2.3.2 Bank.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Bankeintrag in die Datei. Die BankNr dieses Objekts darf noch nicht von einem anderen Eintrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Bank, Schreiben

2.3.3 Bank.Lesen

Integer Lesen *(BankNr)*

Liest einen 'Interne Banken' Datensatz von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *BankNr*: String[5]

Der Kürzel des Bankeintrags.

Siehe Bank, BankNr

2.3.4 Bank.LesenRel

Integer LesenRel *(Wie)*

Liest einen Bankeintrag von der Datei relativ zum Eintrag, die sich jetzt im Bank-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Bankeintrag gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Bank, Lesen

2.3.5 Bank.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Bankeintrag, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Bank, Einfuegen

2.4 BankInfo Objekt

Bezeichnung Mit diesem Objekt können die Bankeverbindungen von DebiNT und KrediNT bearbeitet werden. Pro PK können mehrere Bankverbindungen festgelegt werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(21) erstellt.

Eigenschaften BankId: String[13]

Der ID einer Bank aus dem **Bankstamm**.

BankKonto: String[25]

Die Konto-Nr des Bankkontos.

BVId: String[5] (Index)

Der eindeutige Kürzel der die Bankverbindung in diesem Objekt identifiziert.

IBAN: String[35]

Die IBAN Nummer.

PkIndex: String[17]

Index für OCR, ESR-Nummer

PKNr: Long

Die PK-Nr des P-Kontos zu der diese Bankverbindung gehört.

Zahlungsart: Integer

Die Zahlungsart dieser Bankverbindung. Für Debitoren sind die Zahlungsarten wie folgt:

0 - Manuell

1 - LSV

2 - IBAN/IPI

3 – LSV+

Für Kreditoren sind folgende Codes definiert:

0 - Manuell

1 - ESR 15-stellig

2 - ESR 16-stellig

3 - ESR 27-stellig

4 - "Roter" (Bank)

5 - "Roter" (Post)

6 - "Roter" (Treuhand)

7 - Postmandat

8 - Fremdwhrung

9 - IBAN/IPI

2.4.1 BankInfo.Einfuegen

Integer Einfuegen()

Schreibt eine neue Bankverbindung in die Datei. Die BVId dieses Objekts darf noch nicht von einem anderen Eintrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe BankInfo, Schreiben

2.4.2 BankInfo.Lesen

Integer Lesen *(BVId)*

Liest einen Bankverbindungs-Datensatz von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *PKNr*: Long

Die PK-Nr des P-Kontos zu der diese Bankverbindung gehört.

BVId: String[13]

Der Kürzel der Bankverbindung.

Siehe BankInfo, BVId

2.4.3 BankInfo.LesenRel

Integer LesenRel *(Wie)*

Liest eine Bankverbindung von der Datei relativ zum Eintrag, der sich jetzt im BankInfo-Objekt befindet. Wenn eine PKNr festgelegt wird, bevor diese Funktion aufgerufen wird, werden nur Bankverbindungen vom angegebenen PK zurückgegeben.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Bankverbindung gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei, resp des PKs.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Beispiel Das Beispiel liest alle Bankverbindungen vom PK 10000 und zeigt sie in einer Dialog Box.Es wird angenommen, dass das Objekt Man als DebiNT- oder KrediNT-Mandantgeöffnet wurde.

```
Dim BankInfo As Object
Set BankInfo = Man.NeuObjekt (21)
BankInfo.PKNr = 10000
res% = BankInfo.LesenRel(0)
Do While res% = 0
MsgBox "ID: " + BankInfo.BVId + Chr$(10) + "IBAN: " + BankInfo.IBAN
res% = BankInfo.LesenRel(3)
Loop
```

Siehe BankInfo, Lesen

2.4.4 BankInfo.Loeschen

Integer Loeschen()

Löscht den zuvor gelesenen Bankeintrag aus der Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe BankInfo, Einfuegen

2.4.5 BankInfo.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Bankeintrag, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe BankInfo, Einfuegen

2.5 Bankstamm Objekt

Bezeichnung Mit diesem Objekt können die Banken vom Bankenstamm aus DebiNT und KrediNT bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(14) erstellt.

Eigenschaften **BankId**: String[13] (Index)

Der eindeutige ID der die Bank in diesem Objekt identifiziert.

BankIdHQ: String[13]

Der eindeutige ID (Clearing-Nr) des Hauptsitzes dieser Bank.

BankKonto: String[25]

Die Konto-Nr des Bankkontos.

Text1: String[33]

Die Adresse der Bank.

Text2: String[33]

Die Adresse der Bank.

Text3: String[33]

Die Adresse der Bank.

Text4: String[33]

Die Adresse der Bank.

Swift: String[13]

Die SWIFT-Code dieser Bank.

Siehe Bank, Mandant

2.5.1 Bankstamm.Browser

Integer Browser *(BankId)*

Mit dieser Methode kann der Benutzer mittels Browsers eine Bank aus dem Bankenstamm suchen und lesen.

Rückgabe 0 wenn ein Datensatz erfolgreich gelesen wurde, sonst einen Fehlerstatus.

Parameter *BankId*: String[13]

Auf diese Bank wird im Browser gescrollt, wenn das Fenster sich öffnet. So steht der Benutzer gleich in der Nähe der Bank die ihn interessiert.

Beispiel Das Beispiel erstellt ein Bankstamm-Objekt und öffnet den Browser mit Bank '733' als Vorschlag:

```
Dim Bankstamm As Object

Set Bankstamm = Man.NeuObjekt(14)

res = Bankstamm.Browser("733")

If (res <> 0) Then

MsgBox "Der Benutzer hat abgebrochen."

Exit Sub

End If
```

Siehe Bankstamm, Lesen, LesenRel

2.5.2 Bankstamm.Einfuegen

Integer Einfuegen()

Schreibt eine neue Bank in die Datei. Der BankId dieses Objekts darf noch nicht von einem anderen Eintrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Bankstamm, Schreiben

2.5.3 Bankstamm.Lesen

Integer Lesen *(BankId)*

Liest einen 'Banken' Datensatz von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *BankId*: String[13]

Der ID der Bank. Meistens ist das die 'Clearing-Nr' der Bank.

Siehe Bankstamm, BankId

2.5.4 Bankstamm.LesenRel

Integer LesenRel *(Wie)*

Liest eine Bank von der Datei relativ zum Eintrag, die sich jetzt im Bankstamm-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Bankeintrag gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (PREV): Liest den vorhergehende Datensatz.

3 (NEXT): Liest den nächste Datensatz.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei.

Siehe Bankstamm, Lesen

2.5.5 Bankstamm.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Bankeintrag, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Bankstamm, Einfuegen

2.6 Beleg

2.6.1 Beleg Objekt

Bezeichnung Mit diesem Objekt können FibuNT Belege bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuBeleg erstellt, es ist aber nicht mit dem Mandant verbunden. Das heisst, Sie können mit irgend einem Mandant- Objekt ein Beleg-Objekt erstellen und mit einem anderen Mandanten den Beleg schreiben.

Eigenschaften LastUserChanged: String[29]

Der Inhalt dieses Properties wird beim Schreiben des Belegs verwendet, um das LastUserChanged Property der einzelnen Buchungen zu setzen. Ist der String leer, so wird der Benutzername verwendet, der zuvor beim Aufruf der SetSecurity Funktion des Mandanten Objekts angegeben wurde. Wurde diese Funktion nie aufgerufen, so wird ein technischer Pseudo-Benutzername verwendet, um LastUserChanged der Buchungen zu setzen und kenntlich zu machen, dass die Buchungen über FibuSDK geschrieben wurden.

Vorerfassen: Boolean

Handelt es sich um eine Vorerfassung?

Wichtig: Es wird vorausgesetzt, dass vorgängig ein „vorerfasster“ Kreditoren-OP erfasst wird. Siehe Property „Vorerfassen“ in OP Objekt.

2.6.2 Beleg.AddEmptyBuchung

Diese Funktion ist für den internen Gebrauch reserviert.

2.6.3 Beleg.AnzBuchungen

Integer AnzBuchungen()

Diese Funktion gibt die Anzahl Buchungen im Beleg zurück.

Rückgabe Anzahl Buchungen in diesem Beleg-Objekt.

Siehe IstLeer

2.6.4 Beleg.Bilanz

Currency Bilanz()

Diese Funktion errechnet die Bilanz des Beleges, d.h. Soll- minus Habenbuchungen. Damit der Beleg ausgeglichen und gespeichert werden kann, muss die Differenz 0 sein.

Rückgabe Die Bilanz der Soll und Haben Buchungen im Beleg.

Siehe Beleg

2.6.5 Beleg.Buchen

Integer Buchen (Datum, IstHaben, KtoID, Text, Betrag)

Diese Funktion speichert eine einfache Buchung in dieses Beleg-Objekt. Es entstehen noch keine Aenderungen am FibuNT Datenbestand. Diese erfolgen erst, wenn der Beleg geschrieben wird.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter Datum: Date

Das Datum der Buchung.

IstHaben: Boolean

Wenn dieser Paramter 'True' ist, wird eine Haben-Buchung gemacht, sonst eine Soll-Buchung

KtoID: String[13]

Die Konto-Nr eines FibuNT-Kontos. Wenn vor dieser Funktion die Funktion **BuchungAufOP** verwendet wurde, wird dieser Parameter nicht benützt, sondern das Fibu-Konto des OPs.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts müssen durch ein Line-Feed 'CHR$(10)' getrennt werden.

Betrag: Currency

Der Netto-Betrag der Buchung. Dieser Betrag darf nicht 0.0 sein.

Beispiel Siehe das Beispiel in BuchenSt.

Siehe Beleg, BuchenSt, BuchenKst

2.6.6 Beleg.Buchen2

Integer Buchen2 (Datum, IstHaben, KtoID, KstID, Text, Betrag, BetragFW)

Diese Funktion speichert eine normale Buchung in dieses Beleg-Objekt. Optional wird eine Kostenstellen-Buchung und/oder eine automatische Steuer-Buchung gemacht.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter Datum: Date

Das Datum der Buchung.

IstHaben: Boolean

Wenn dieser Paramter 'True' ist, wird eine Haben-Buchung gemacht, sonst eine Soll-Buchung

KtoID: String[13]

Die Konto-Nr eines FibuNT-Kontos. Wenn vor dieser Funktion die Funktion BuchungAufOP verwendet wurde, wird dieser Parameter nicht benützt, sondern das Fibu-Konto des OPs.

KstID: String[13]

Die Konto-Nr einer Kostenstelle oder ein leerer String, wenn keine

Kostenstellen-Buchung gemacht werden soll.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts müssen durch ein Line-Feed 'CHR$(10)' getrennt werden.

Betrag: Currency

Der Netto-Betrag der Buchung. Dieser Betrag darf nicht 0.0 sein.

BetragFW: Currency

Der Netto-Betrag der Buchung in der Fremdwährung. Dieser Betrag muss 0.0 sein, wenn KtoID kein Fremdwährungskonto ist.

Beispiel Siehe das Beispiel in BuchenSt.

Siehe Beleg, BuchenKst

2.6.7 Beleg.BuchenKst

Integer BuchenKst (Datum, IstHaben, Kostenstelle, Kostenart, Text, Betrag)

Diese Funktion speichert eine reine Kostenstellen-Buchung in dieses Beleg-Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter Datum: Date

Das Datum der Buchung.

IstHaben: Boolean

Wenn dieser Paramter 'True' ist, wird eine Haben-Buchung gemacht, sonst eine Soll-Buchung

Kostenstelle: String[13]

Die Konto-Nr einer Kostenstelle.

Kostenart: String[13]

Die Konto-Nr eines Fibukontos.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts müssen durch ein Line-Feed 'CHR$(10)' getrennt werden.

Betrag: Currency

Der Netto-Betrag der Buchung. Dieser Betrag darf nicht 0.0 sein.

Siehe Buchen2

2.6.8 Beleg.BuchenKst2

Integer BuchenKst2 (Datum, IstHaben, KtoID, KstID, KstID2, Text, Betrag, BetragFW, Steuerbetrag,
Steuer, Modus)

Diese Funktion macht fast dasselbe wie die Funktion BuchenSt. Der Unterschied ist, dass hier zwei Kostenstellen angegeben werden können (zum Beispiel KST und KTR). Es werden dann zwei Kostenstellenbuchungen erzeugt.

Wichtig: Der Parameter KstID2 darf bis auf weiteres nicht mehr verwendet werden. Dies ist auf einen Fehler zurückzuführen, der mit der Einführung von Verteilmethoden entstanden ist. Der Fehler wird in einer zukünftigen Version von FibuSDK korrigiert werden, ab dann kann KstID2 voraussichtlich wieder verwendet werden.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus. keine.

Parameter *Datum*: Date

Das Datum der Buchung.

IstHaben: Boolean

Wenn dieser Paramter 'True' ist, wird eine Haben-Buchung gemacht, sonst eine Soll-Buchung

KtoID: String[13]

Die Konto-Nr eines FibuNT-Kontos. Wenn vor dieser Funktion die Funktion BuchungAufOP verwendet wurde, wird dieser Parameter nicht benützt, sondern das Fibu-Konto des OPs.

KstID: String[13]

Die Konto-Nr einer Kostenstelle oder ein leerer String, wenn keine

Kostenstellen-Buchung gemacht werden soll.

KstID2: String[13]

Die Konto-Nr einer zweiten Kostenstelle oder ein leerer String, wenn keine zweite Kostenstellen-Buchung gemacht werden soll.

Wichtig: Dieser Parameter darf bis auf weiteres nicht mehr verwendet werden. Siehe die Methoden-Dokumentation.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts müssen durch ein Line-Feed 'CHR$(10)' getrennt werden.

Betrag: Currency

Der Netto-Betrag der Buchung. Dieser Betrag darf nicht 0.0 sein.

BetragFW: Currency

Der Netto-Betrag der Buchung in der Fremdwährung. Dieser Betrag muss 0.0 sein, wenn KtoID kein Fremdwährungskonto ist.

Steuerbetrag: Currency

Der Steuerbetrag der Buchung. Dieser Betrag wird nur benützt, wenn der Parameter Steuer angegeben wird.

Steuer: Steuer

Das Steuer-Objekt muss einen gültigen Steuersatz enthalten.

Modus: Integer

0=Einfügen, 1=Überschreiben, 2=Ignorieren

Kommentar Mit dieser Funktion kann auch eine reine Steuerbuchung gemacht werden. Das ist der Fall, wenn KtoID das selbe Konto wie im Steuersatz ist. Auch für diese Art von Buchung muss der Netto-Betrag angegeben werden, damit die FibuNT Steuerauswertungen richtig funktionieren.

Falls der Modus 0 ist, wird der OP versucht zu archivieren. Falls er schon archiviert ist, wird ein Fehlercode zurückgegeben. Danach kann ggf. der Modus auf 1 gesetzt werden und die Buchen-Methode nochmals aufgerufen werden. Dann wird der archivierte OP überschrieben. Mit dem Parameter 2 wird versucht den OP einzufügen, wenn das misslingt, wird der Beleg trotzdem gescheiterter Archivierung gespeichert, sofern die Archivierung nicht auf zwingend eingestellt ist. Beachten Sie bitte, dass ein OP nur archiviert werden kann, wenn mit SetBarcode ein Barcode zugeordnet wurde.

Beispiel Das Beispiel zeigt wie eine Soll-Buchung mit KST und KTR erstellt wird:

```
' Beleg-Objekt erstellen
Dim datum, Beleg As Object
Set Beleg = Mandant.NeuBeleg()
datum = Now()
betrag@ = 250

' Die Soll Buchung mit Kostenstelle
res% = Beleg.BuchenKst2(datum, False, "3000", "9001", "9001", "buchtext",

betrag@, 0, 0, Nothing)
If res% <> 0 GoTo ShowErr

' Die Haben Buchung
res% = Beleg.Buchen(datum, True, "1020", "buchtext", betrag@) If res% <> 0 GoTo
ShowErr

' Beleg schreiben

Beleg.SetBelegNr (Mandant.FreieBelegNr(10000))
res% = Mandant.BelegEinfuegen(Beleg)
If res% <> 0 GoTo ShowErr

MsgBox "Beleg wurde geschrieben", MB_OK
Exit Sub

ShowErr:
MsgBox "Fehler beim Buchen", MB_OK
Exit Sub
```

Siehe Beleg, BuchenSt, Buchen, Mandant.BelegEinfuegen

Beleg, Einfuegen, BuchungAufOp, SetBarcode

2.6.9 Beleg.BuchenSt

Integer BuchenSt (Datum, IstHaben, KtoID, KstID, Text, Betrag, BetragFW, Steuerbetrag, Steuer)

Diese Funktion speichert eine normale Buchung in das Beleg-Objekt; optional wird eine Kostenstellen-Buchung und/oder eine automatische Steuer-Buchung gemacht.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter Datum: Date

Das Datum der Buchung.

IstHaben: Boolean

Wenn dieser Paramter 'True' ist, wird eine Haben-Buchung gemacht, sonst eine Soll-Buchung

KtoID: String[13]

Die Konto-Nr eines FibuNT-Kontos. Wenn vor dieser Funktion die Funktion BuchungAufOP verwendet wurde, wird dieser Parameter nicht benützt, sondern das Fibu-Konto des OPs.

KstID: String[13]

Die Konto-Nr einer Kostenstelle oder ein leerer String, wenn keine

Kostenstellen-Buchung gemacht werden soll.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts müssen durch ein Line-Feed 'CHR$(10)' getrennt werden.

Betrag: Currency

Der Netto-Betrag der Buchung. Dieser Betrag darf nicht 0.0 sein.

BetragFW: Currency

Der Netto-Betrag der Buchung in der Fremdwährung. Dieser Betrag muss 0.0 sein, wenn KtoID kein Fremdwährungskonto ist.

Steuerbetrag: Currency

Der Steuerbetrag der Buchung. Dieser Betrag wird nur benützt, wenn der Parameter Steuer angegeben wird.

Steuer: Steuer

Das Steuer-Objekt muss einen gültigen Steuersatz enthalten.

Kommentar Mit dieser Funktion kann auch eine reine Steuerbuchung gemacht werden. Das ist der Fall, wenn KtoID dasselbe Konto wie im Steuersatz ist. Auch für diese Art von Buchung muss der Netto-Betrag angegeben werden, damit die FibuNT Steuerauswertungen richtig funktionieren.

Beispiel Das Beispiel zeigt wie eine Soll- und Haben-Buchung in einen Beleg gespeichert wird und dieser anschliessend geschrieben wird:

```
' Beleg-Objekt erstellen
Dim datum, Beleg As Object
Set Beleg = Mandant.NeuBeleg()
datum = Now()
betrag@ = 250

' Die Soll Buchung mit Kostenstelle
res% = Beleg.Buchen2(datum, False, "3000", "9002", "buchtext", betrag@, 0)
If res% <> 0 GoTo ShowErr

' Die Haben Buchung
res% = Beleg.Buchen(datum, True, "1020", "buchtext", betrag@) If res% <> 0 GoTo
ShowErr

' Beleg schreiben
Beleg.SetBelegNr (Mandant.FreieBelegNr(10000))
res% = Mandant.BelegEinfuegen(Beleg)
If res% <> 0 GoTo ShowErr

MsgBox "Beleg wurde geschrieben", MB_OK
Exit Sub

ShowErr:
MsgBox "Fehler beim Buchen", MB_OK
Exit Sub
```

Siehe Beleg, BuchenKst, BuchenKst2, Buchen, Mandant.BelegEinfuegen

2.6.10 Beleg.Buchung

Buchung Buchung *(Index)*

Mit dieser Funktion können die Buchungen in einem Beleg einzeln bearbeitet werden.

Rückgabe Ein Buchung-Objekt, oder NULL wenn der Index ungülig ist.

Parameter *Index*: Integer

Der Index der gewünschten Buchung. Der Index ist 'zero-based'. Das bedeutet, die erste Buchung ist 0, die zweite 1, usw.

Kommentar Nicht alle Funktionen des Buchungs-Objekts können verwendet werden. z.B Die Lese- und Schreibfunktionen.

Das Buchungs-Objekt darf mit dem Set Befehl in eine VB-Variable gespeichert werden, um die verschiedenen Eigenschaften der Buchung zu lesen oder ändern. Aber nach einem Aufruf der Buchen, Leeren, oder eine andere Funktion, die die Anzahl der Buchungen im Beleg verändert, darf die Variable nicht mehr benützt werden. Sie muss neu gesetzt werden. Das gilt auch wenn die Anzahl der Buchungen grösser wird.

Beispiel Die erste Buchung eines Belegs ändern

```
Dim Buchung As Object

Set Buchung = Beleg.Buchung(0)

Buchung.KontoNr = "1020"

Buchung.GKontoNr = "3000"

'oder einfach

Beleg.Buchung(0).KontoNr = "1020"
```

Siehe Beleg, Mandant.BelegEinfuegen

2.6.11 Beleg.BuchungAufOP

Integer BuchungAufOP *(Man, OPNr, ErsteBOP)*

Mit dieser Funktion legen Sie fest, dass die nächste Buchung auf ein OP gebucht werden soll. Der OP muss im angegebenen Mandanten schon vorhanden sein. Um einen neuen OP zu erstellen, sehen Sie das Beispiel unter OP.Einfuegen. Der OP und dessen Personenkonto werden mit dieser Funktion gelesen, um die benötigen Informationen zu erhalten.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Man*: Mandant

Das Mandant-Objekt mit dem dieser Beleg, wenn er fertig ist, geschrieben wird.

OPNr: String[13]

Das OP auf das gebucht werden soll.

ErsteBOP: Boolean

Wenn dieser Paramter 'True' ist, wird diese Buchung als die OP-Entstehungsbuchung behandelt. Sie legt den Rechnungsbetrag des OPs fest.

Kommentar Diese Funktion selber schreibt keine Daten in die Buchung, sondern die darauf folgende Buchen Funktion wird die Angaben dieser Funktion verwenden um eine OP-Buchung zu machen.

Siehe Beleg, Buchen, Buchen2, BuchenSt

2.6.12 Beleg.Einfuegen

Integer Einfuegen ***(Man)***

Schreibt einen neuen Beleg in die Fibu. Der Beleg muss ausgeglichen und darf nicht leer sein.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Man*: Mandant

Ein gültiges Mandant-Objekt, mit dem schon ein Mandant.Login gemacht wurde.

Siehe Beleg, Ersetzen, Mandant.NeuBeleg, Lesen

2.6.13 Beleg.Einfuegen2

Diese Funktion ist für den internen Gebrauch reserviert.

2.6.14 Beleg.Ersetzen

Integer Ersetzen *(Man)*

Ähnlich wie Einfuegen schreibt diese Funktion einen Beleg in die Fibu. Sie löscht aber zuerst den Beleg mit der gleichen Beleg-Nr in der Datenbank. Der Beleg muss ausgeglichen und darf nicht leer sein.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Man*: Mandant

Ein gültiges Mandant-Objekt, mit dem schon ein Mandant.Login gemacht wurde.

Beispiel Das Beispiel zeigt wie ein Beleg gelesen, verändert und dieser anschliessend geschrieben wird. Keine Fehlerüberprüfung:

```
' Mandant erstellen und öffnen
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")
Man.Login 1, "C:\ProgramData\Sage\Data\Rewe\SageDemo16"

' Beleg Objekt erstellen und BelegNr 1 lesen
Dim Beleg As Object
Set Beleg = Man.Neubeleg()
res% = Beleg.Lesen(1, Man)

' Bei allen Buchungen den Code setzen
i% = 0
Do While i% < Beleg.AnzBuchungen
Beleg.Buchung(i%).Code = "Spezial"
i% = i% + 1
Loop

' Beleg schreiben
res% = Beleg.Ersetzen(Man)
```

Siehe Beleg, Mandant.NeuBeleg, Lesen

2.6.15 Beleg.Ersetzen2

Diese Funktion ist für den internen Gebrauch reserviert.

2.6.16 Beleg.IstLeer

Boolean IstLeer()

Mit dieser Funktion können Sie feststellen, ob schon Buchungen in diesem Beleg vorhanden sind.

Rückgabe keine.

Siehe Beleg, AnzBuchungen

2.6.17 Beleg.Leeren

void Leeren()

Diese Funktion entfernt alle Buchungen aus dem Beleg.

Rückgabe keine

Kommentar Ein Beleg ist leer, wenn er mit der Funktion Mandant.NeuBeleg frisch erstellt wurde.

Siehe AnzBuchungen

2.6.18 Beleg.Lesen

Integer Lesen *(BelegNr, Man)*

Liest einen ganzen Beleg aus der Buchungs-Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *BelegNr*: long

Die BelegNr des Belegs, der gelesen werden soll.

Man: Mandant

Ein gültiges Mandant-Objekt, mit dem schon ein Mandant.Login gemacht wurde.

Siehe Beleg, Buchung.BelegNr, Mandant.BelegEinfuegen

2.6.19 Beleg.RemoveBuchung

Diese Funktion ist für den internen Gebrauch reserviert.

2.6.20 Beleg.SetBarcode

void SetBarcode*(Barcode)*

Diese Funktion setzt den Barcode der beim Archivieren eines OPs als Schlüssel verwendet werden soll. Der Barcode sollte unmittelbar vor dem Erstellen des OPs mit BuchungAufOp und Einfuegen gesetzt werden. Vergleiche auch SetArchivModus.

Rückgabe keine.

Parameter *Barcode*: String

Der Barcode

Siehe Beleg, Einfuegen, BuchungAufOp, SetArchivModus

2.6.21 Beleg.SetBelegNr

void SetBelegNr *(Beleg-Nr)*

Diese Funktion legt die Beleg-Nr für alle Buchungen fest, die in diesem Beleg schon vorhanden sind. Diese Funktion sollte in dieser Version erst benützt werden, wenn der Beleg bereits alle Buchungen enthält, also kurz vor dem Schreiben.

Rückgabe keine

Parameter *Beleg-Nr*: Long

Die Beleg-Nr mit dem dieser Beleg gespeichert werden soll.

Siehe Beleg, Buchen, Mandant.BelegEinfuegen, Mandant.FreieBelegNr

2.6.22 Beleg.SetBookingBlockedCheck

void SetBookingBlockedCheck *(bCheck, bSilent)*

Mit dieser Funktion kann die Überprüfung, ob eine Buchungssperre aktiv ist, ein- oder ausgeschaltet werden. Per Default ist die Prüfung aktiv, d.h. beim Schreiben eines Belegs wird geprüft, ob das Datum der einzelnen Buchungen in eine Periode fällt, für die eine Buchungssperre definiert wurde. Fällt das Datum auf eine Buchungssperre, so kann der Beleg nicht geschrieben werden; per Default wird dann auch eine Fehlermeldung am Bildschirm angezeigt. Das Anzeigen der Fehlermeldung kann unterdrückt werden.

Rückgabe keine

Parameter *bCheck* Boolean

Wenn dieser Parameter 'False' ist wird die Buchungssperre-Überprüfung deaktiviert.

bSilent: Boolean

Wenn dieser Parameter 'True' ist wird das Anzeigen der Fehlermeldung unterdrückt.

2.6.23 Beleg.SetGrpCode

void SetGrpCode *(Grp-Code)*

Diese Funktion legt den Gruppen-Code für alle Buchungen fest, die in diesem Beleg sind. Diese Funktion sollte in dieser Version erst benützt werden, wenn der Beleg bereits alle Buchungen enthält (somit kurz vor dem Schreiben).

Rückgabe keine

Parameter *Grp-Code*: Integer

Der Gruppencode.

Siehe Beleg, Mandant.BelegEinfuegen

2.6.24 Beleg.Stornieren

void Stornieren()

Diese Funktion setzt den Buchung.Typ aller Buchungen im diesem Beleg auf das Gegenteil.

Kommentar Mit Einfuegen kann der bestehende Beleg verändert werden. Die Beleg-Nummber (SetBelegNr) kann neu gesetzt werden und mit Einfuegen wird ein neuer Beleg erstellt.

Achtung: Das Buchung.IstErsteBOP Flag wird bei allen OP-Buchungen, auf False gesetzt.

Siehe Beleg, Lesen

2.7 BuchArchiv Objekt

Bezeichnung Mit diesem Objekt können Buchungen im Archiv gelesen werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt erstellt.

Eigenschaften BelegNr: Long

Die Belegnummer der Buchung im Archiv.

Diese Eigenschaft kann verwendet werden, um den Index zu initialisieren, wenn die Methode BuchArchiv.LesenRel verwendet werden soll.

ArchivNr: Long

Die Archivnummer der Buchung im Archiv. Alle Buchungen im Archiv sind durchnummeriert.

Diese Eigenschaft kann verwendet werden, um den Index zu initialisieren, wenn die Methode BuchArchiv.LesenRel verwendet werden soll.

TimeStamp: Date

Das Datum der Löschung.

Diese Eigenschaft kann verwendet werden, um den Index zu initialisieren, wenn die Methode BuchArchiv.LesenRel verwendet werden soll.

2.7.1 BuchArchiv.LesenRel

Integer LesenRel *(Wie)*

Liest eine Buchung aus dem Archiv on das BuchArchiv-Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Datensatz gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (PREV): Liest den vorhergehende Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz (welcher ein höheres Datum hat, als mit TimeStamp gesetzt wurde), mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

2.8 Buchung Objekt

Bezeichnung Mit diesem Objekt können Buchungen eines Mandanten einzeln bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuBuchung erstellt.

Eigenschaften BelegNr: Long

Die Belegnummer dieser Buchung. Alle Buchungen mit der gleichen Belegnummer bilden einen Beleg. Diese Eigenschaft sollte nur geändert werden, um mit der Methode Buchung.LesenRel den richtigen Datensatz zu lesen.

Achtung! Wenn diese Eigenschaft gesetzt wird, wird automatisch die Eigenschaft Buchung.BelegIndex auf den Wert 0 gesetzt.

BelegIndex: Integer

Der Index der Buchungen innerhalb des Belegs. Die erste Buchung hat den Index 0, die zweite hat Index 1, usw. Der 'BelegIndex' hat nichts mit der Methode Buchung.SetIndex zu tun. Diese Eigenschaft sollte nur geändert werden, um mit der Methode Buchung.LesenRel den richtigen Datensatz zu lesen.

Betrag: Currency

Der Betrag dieser Buchung in der Leitwährung. Dieser Betrag ist immer der Netto-Betrag, d.h. ohne den Steuerbetrag.

BetragFW: Currency

Der Betrag dieser Buchung in der Fremdwährung.

BetragSteuer: Currency

Der Steuerbetrag dieser Buchung.

Buchmaske: Integer

Die Identifikationsnummer der Buchungsmaske, beziehungsweise des Dialogs mit dem dieser Beleg bearbeitet werden soll.

CalculatedAmount: Currency

Diese Eigenschaft ist für den internen Gebrauch reserviert.

CalculatedKonto: String[13]

Diese Eigenschaft ist für den internen Gebrauch reserviert.

Code: String[11]

Der frei definierbare Code des Kontos.

Datum: Date

Das Datum der Buchung.

GFNr: Long

Die GFNr (Geschäftsfall-Nr) der Buchung.

GKontoNr: String[13]

Auf dieses Gegenkonto bezieht sich die Buchung.

Gruppe : Integer

Der Gruppen-Code.

IstErsteBOP: Boolean

Diese Eigenschaft ist 'True' für die Entstehungsbuchung eines OPs. Pro OP darf es nur eine Buchung geben, die diese Eigenschaft auf True gesetzt hat.

IstFixiert: Boolean (nur Lesen)

Wenn diese Eigenschaft 'True' ist, kann die Buchung nicht verändert werden, weil sie schon 'fixiert', 'journalisiert' oder gesperrt wurde.

IstFwBuchung: Boolean (nur Lesen)

Wenn dieser Schalter den Wert 'True' hat, handelt es sich um eine Fremdwährungsbuchung.

IstGesperrt: Boolean

Wenn diese Eigenschaft 'True' ist, kann die Buchung nicht verändert werden. Eine Buchung wird von FibuNT gesperrt, wenn sie beim Treuhand ist.

IstHaben: Boolean

Diese Eigenschaft ist 'True', wenn der Betrag der Buchung auf der Haben-Seite ist.

KontoNr: String[13]

Diese Buchung bezieht sich auf diese Konto.

KstIndex: Integer

Der Index der Kostenstellen-Buchung der mit dieser Buchung verbunden ist. Der Index ist '1 based'. Das heist 1 ist die erste Buchung, 2 die zweite, usw. Wenn diese Eigenschaft 0 zurück gibt, ist keine KST- Buchung mit dieser verbunden.

Beim setzen dieser Eigenschaft wird nicht überprüft ob der Wert innerhalb des Belegs liegt. Stellen Sie entsprechend sicher, dass KstIndex entweder 0 ist, oder zwischen 1 und Anzahl-Buchungen im Beleg liegt.

LastUserChanged: String[29] (nur Lesen)

Der Name des Benutzers, der diese Buchung zuletzt verändert hat. Diese Eigenschaft kann nicht direkt verändert werden, sondern wird beim Schreiben eines Belegs für alle Buchungen des Belegs auf den gleichen Wert gesetzt. Siehe Eigenschaft LastUserChanged beim Beleg Objekt.

OPNr: String[13]

Die OP-Nr dieser Buchung.

PKNr: Long

Die PK-Nr dieser Buchung.

SteuerID: String[5]

Der Kürzel eines Steuersatzes mit dem der Steuerbetrag errechnet wurde.

SteuerIndex: Integer

Der Index der Steuer-Buchung der mit diese Buchung verbunden ist. Der Index ist '1 based'. Das heisst 1 ist die erste Buchung, 2 die zweite, usw. Wenn diese Eigenschaft 0 zurück gibt, ist keine KST-Buchung mit dieser verbunden.

SteuerNr: Integer (nur Lesen)

Die Nummer der MWst-Auswertung die diese Buchung beinhaltet, oder in der diese Buchung mitgerechnet wurde.

Text: String[61]

Der Buchungstext. Die zwei Zeilen des Buchungstexts werden durch ein Line-Feed 'CHR$(10)' getrennt.

DocID: String[15]

Die Dokument-ID verbindet die Buchung mit einem Beleg in einem Archiv.

TimeStamp: Date

Das Datum der letzten Änderung. Diese Eigenschaft ist "read-only", sie kann jedoch verwendet werden, um den Index zu initialisieren, wenn die Methode Buchung.SetIndex mit Index-Nr 7 verwendet werden soll.

Typ: Integer

Der Typ der Buchung. Folgende Typen sind definiert:

0 - Normale Buchung

1 - Kostenstellen Buchung

2 - Steuer-Buchung

TypTrans: Integer

Diese Eigenschaft bestimmt ob eine Buchung eine Transitorische- Buchung ist, und was für eine. Alle Buchungen die zum gleichen Beleg gehören sollten bei dieser Eigenschaft den gleichen Wert haben. Folgende Typen sind definiert:

0 - Keine Transitorische-Buchung

1 - Transitorische Einbuchung

2 - Transitorische Ausbuchung

Ursprung: Integer

Der Ursprung der Buchung. Folgende Typen sind definiert:

0 - FibuNT,

1 - DebiNT,

2 - KrediNT

3 - Andere

2.8.1 Buchung.Lesen

Integer Lesen (BelegNr, Index)

Liest eine einzelne Buchung aus der Buchungs-Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *BelegNr*: Long

Die BelegNr der Buchung, die gelesen werden soll.

Index: Integer

Der Index der Buchung.

Siehe Buchung, BelegNr, LesenRel

2.8.2 Buchung.LesenRel

Integer LesenRel *(Wie)*

Relativ zur Buchung, die sich jetzt schon in diesem Buchung-Objekt befindet, wird eine Buchung aus der Datei gelesen.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methoden zum Lesen des nächsten Kontos:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz noch einmal, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

4 (GREATEROREQUAL): Liest den gleichen Datensatz noch einmal, oder den nächsten Datensatz, gemäss Primärschlüssel, mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zum vorhandenen Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Buchung, Lesen, SetIndex

2.8.3 Buchung.SetIndex

Integer SetIndex *(Code)*

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code*: Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach Beleg-Nr.

1 : Sortiert nach Konto/Datum.

2 : Sortiert nach PK-Nr/Datum.

3 : Sortiert nach OP-Nr/Datum.

4 : Sortiert nach Ursprung.

5 : Sortiert nach DocId.

6 : Sortiert nach GFNr/Datum.

7 : Sortiert nach Stampdatum.

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

Beispiel Das Beispiel öffnet einen Mandant und liest alle Buchungen die auf OP "60001" gebucht wurden und zeigt sie in einer Message Box:

```
Dim man As Object

' Mandant im Debitoren-Modus öffnen
Set man = CreateObject("FibuNT.Mandant")
res% = man.Login(2, "C:\Proj\Outback\Data\Rewe\Westlox2007")

' Neues Buchungs-Objekt holen
Dim buch As Object
Set buch = man.NeuBuchung()

' Index auf OP-Nr setzen und initialisieren
Dim SuchOP As String
SuchOP = "60001"
buch.SetIndex (3)
buch.OPNr = SuchOP

' Jetzt werden ab der angegebenen OP-Nr Buchungen gelesen
' Im While wird getestet ob noch eine Buchung vorhanden ist und ob die Buchung zum gesuchten OP gehört.
While (buch.LesenRel(3) = 0 And buch.OPNr = SuchOP)
MsgBox ("Beleg: " & buch.belegNr & "(" & buch.BelegIndex & ")
```

```
- " & buch.Text)
Wend
```

Siehe Buchung, LesenRel

2.8.4 Buchung.GetNoFwKonto

Boolean GetNoFwKonto()

Gibt true zurück, falls diese Buchung zwar Fremdwährungsbeträge aufweisen kann, aber trotzdem auf ein Leitwährungskonto gebucht wird.

2.8.5 Buchung.SetNoFwKonto

void SetNoFwKonto()

Setzt ein Flag, das besagt, dass diese Buchung zwar Fremdwährungsbeträge aufweisen kann, aber trotzdem auf ein Leitwährungskonto gebucht wird.

2.8.6 Buchung. CalcAmounts

Diese Funktion ist für den internen Gebrauch reserviert.

2.8.7 Buchung. GetNextCalcData

Diese Funktion ist für den internen Gebrauch reserviert.

2.9 Budget

2.9.1 Budget Objekt

Bezeichnung Mit diesem Objekt können Konten-Budget bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt erstellt.

Eigenschaften BudgetListe: String (nur Lesen)

Gibt eine Liste (Kommagetrennt) der möglichen Budgettypen zurück, die in der Methode Budget.Initialisieren als Parameter angegeben werden können.

Diesem Eigenschaft bezieht sich nicht auf einen bestimmten Budget- Satz, desshalb kann sie auch ohne etwas zu lesen aufgerufen werden.

KontoNr: String[13] (Index)

Die Konto-Nr des Kontos dessen Budget bearbeitet werden soll.

KstNr: String[13] (Index)

Die Kostenstellen-Nr.

2.9.2 Budget.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Budgetsatz (12 Monate) in die Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Kommentar Beide das Konto und die Kostenstelle sollten als Konto vorhanden sind, bevor ein Budget-Betrag eingefügt wird, da sonst der Eintrag vom Rewe her weder gesehen, noch gelöscht werden kann.

Siehe Budget, Schreiben

2.9.3 Budget.GetMonat

Currency GetMonat *(Monat)*

Liest den Budgetbetrag eines bestimmten Monats.

Rückgabe Der Budgetbetrag für den angegebenen Monat.

Parameter *Monat*: Currency

Der Monat (0-11). Monat = 0 ist der Monat in dem das Geschäftsjahr anfängt.

Kommentar Zur Zeit werden nur 12 Monate unterstützt.

Siehe Budget, SetMonat

2.9.4 Budget.GetQuartal

Currency GetQuartal *(Quartal)*

Liest den Budgetbetrag eines bestimmten Monats.

Rückgabe Der Budgetbetrag für das angegebene Quartal.

Parameter *Quartal*: Currency

Das Quartal (0-3). Quartal = 0 sind die ersten drei Monate des Geschäftsjahres anfängt.

Kommentar Zur Zeit werden nur 4 Quartale unterstützt.

Siehe Budget, SetQuartal

2.9.5 Budget.Initialisieren

Integer Initialisieren *(BudgetTyp)*

Initialisiert und öffnet die entsprechende Budget-Dateien.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *BudgetTyp*: Integer

Der gewünschte Typ des Budgets.

1 - Budget für Geschäftsjahr

2 - Budget für Folgejahr

Siehe Budget, Lesen, LesenRel

2.9.6 Budget.Lesen

Integer Lesen (KontoNr, KstNr)

Liest einen Budget-Satz von der Datenbank für ein bestimmtes Konto und Kostenstelle.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *KontoNr*: String[13]

Der Kürzel des Kontos oder Kostenstelle, dessen Budget gelesen werden soll.

KstNr: String[13]

Der Kürzel der Kostenstelle oder des Kostenträgers, dessen Budget gelesen werden soll.

Kommentar Bevor Sie lesen können muss die Methode Initialisieren aufgerufen werden.

Siehe Budget, KontoNr, LesenRel

2.9.7 Budget.LesenRel

Integer LesenRel *(Wie)*

Liest ein Budget von der Datenbank relativ zum Datensatz, der sich jetzt im Budget-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Datensatz gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Budget, Lesen

2.9.8 Budget.Loeschen

Integer Loeschen()

Entfernt den zuvor gelesenen Budgetsatz aus der Datenbank.

Rückgabe 0 wenn erfolgreich gelöscht, sonst einen Fehlerstatus.

Siehe Budget, Lesen

2.9.9 Budget.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Budgetsatz (12 Monate) in die Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Beispiel Das Beispiel Liest Konto 6000, ändert den ersten Monat und speichert den Wert:

```
Dim man As Object
man = CreateObject("FibuNT.Mandant")
man.Login(1, "C:\ProgramData\Sage\Data\Rewe\SageDemo16")

Dim budget As Object
Dim res As Integer
budget = man.NeuObjekt(17)

budget.Initialisieren(1)
budget.Lesen("6000", "")
budget.SetMonat(0, 111.25)
res = budget.Schreiben()
If (res = 0) Then
MsgBox("Konto 6000 erfolgreich geschrieben!")
End If
```

Siehe Budget, Einfuegen

2.9.10 Budget.SetMonat

void SetMonat *(Monat, Wert)*

Setzt den Budgetbetrag eines bestimmten Monats.

Rückgabe keine

Parameter *Monat*: Integer

Der Monat (0-11). Monat = 0 ist der Monat in dem das Geschäftsjahr anfängt.

Wert: Currency

Der Budgetbetrag des Monats.

Kommentar Zur Zeit werden nur 12 Monate unterstützt.

Siehe Budget, GetMonat

2.9.11 Budget.SetQuartal

void SetQuartal *(Quartal, Wert)*

Setzt den Budgetbetrag eines bestimmten Quartals.

Rückgabe keine

Parameter *Quartal*: Integer

Das Quartal (0-3). Quartal = 0 sind die ersten drei Monate des Geschäftsjahres anfängt.

Wert: Currency

Der Budgetbetrag des Quartals. drei Monate des Geschäftsjahres anfängt.

Kommentar Zur Zeit werden nur 4 Quartale unterstützt.

Siehe Budget, GetQuartal

2.10 Dauerauftrag

2.10.1 Dauerauftrag Objekt

Bezeichnung Mit diesem Objekt können Daueraufträge bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(27) erstellt.

Eigenschaften AlsOP: Boolean

Wenn dieses Flag auf True gesetzt wird, wird ein OP erstellt und darauf gebucht.

AuftragsNr: String[5] (Index)

Die eindeutige Nr des Dauerauftrags in diesem Objekt identifiziert.

BankNr: String[5]

Die BankNr.

Betrag: Currency

Der Betrag dieses Dauerauftrags.

DatumAnfang: Date

Anfangsdatum des Auftrags.

DatumEnde: Date

Enddatum des Auftrags.

Empfaenger: String[13]

Der Adress-ID des Empfängers.

GFNr: Long

Die GFNr (Geschäftsfall-Nr) der Buchung.

Periode: Integer

Die Periode mit welcher der Dauerauftrag ausgeführt werden soll. Folgende Perioden sind definiert:

0 - Monatlich

1 - Quartalweise

2 - Halbjährlich

3 - Jährlich

4 - in Tagen

Tage: Long

Anzahl Tage.

Vorlage: Long

Die Buchungsvorlage die verwendet werden soll für diesen Dauerauftrag.

Text: String[61]

Der Text.

Siehe Mandant

2.10.2 Dauerauftrag.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Dauerauftrag in die Datei. Der AuftragsNr dieses Objekts darf noch nicht von einem anderen Dauerauftrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Dauerauftrag, Schreiben

2.10.3 Dauerauftrag.Lesen

Integer Lesen *(AuftragsNr)*

Liest einen Datensatz von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *AuftragsNr*: String[13]

Die Auftrags-Nr eines Dauerauftrags.

Siehe Dauerauftrag, AuftragsNr

2.10.4 Dauerauftrag.LesenRel

Integer LesenRel *(Wie)*

Liest einen Dauerauftrag von der Datei relativ zum Eintrag, der sich jetzt im Dauerauftrag-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Datensatz gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Dauerauftrag, Lesen

2.10.5 Dauerauftrag.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Dauerauftrag, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Dauerauftrag, Einfuegen

2.11 EBanking

2.11.1 EBanking Objekt

Bezeichnung Mit diesem Objekt können Einstellungen bezüglich EBanking abgefragt werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(20) erstellt. Falls kein EBanking für den Mandanten konfiguriert ist, liefert die Funktion NULL zurück.

Eigenschaften Datenpfad: String (nur Lesen)

Pfad, wo globale EBankingdaten gespeichert werden sollen..

Host: String (nur Lesen)

Der Host ist jender Computer, von dem aus eine EBankingverbindung aufgebaut werden kann.

Poolpfad: String (nur Lesen)

Pfad, wohin Dateien geschrieben werden müssen, damit sie als pendente Aufträge verarbeitet werden.

2.11.2 EBanking.DateiInPoolEinfuegen

long DateiInPoolEinfuegen (Datei, Type, SequenzNummer)

Kopiert eine Datei in den Pool der pendenten Aufträge

Rückgabe Für die Datei verwendete Sequenznummer

Parameter *Datei*: String

Originalandatei mit Pfad, die in den Pool hinzugefügt werden soll.

Type: long

Art der Datei. Gültige Werte sind:

SequenzNummer: long

Nummer der Sequenz im PendenzenPool. Sie kann mit NeueSequenzNummer() erstellt werden. Falls mehrere Dateien zu demselben EBanking-Auftrag gehören, muss für DateiInPoolEinfuegen mehrmals aufgerufen werden und immer dieselbe Sequenznummer

muss übergeben werden. Falls 0 übergeben wird, so wird automatisch eine neue Sequenznummer generiert.

1 DebitDirect (senden)

2 BESRVESR (empfangen)

3 DTA (senden)

4 EZAG (senden)

5 MT940/MT950 Bankauszüge (empfangen)

6 YellowBill (senden)

7 PayNet (senden)

2.11.3 EBanking.IstDateitypGueltig

Boolean IstDateitypGueltig *(BcNr, Type)*

Diese Funktion dient zur Überprüfung, ob ein Dateiformat über das EBanking abgewickelt werden kann.

Rückgabe TRUE, wenn die Datei über das EBanking abgewickelt werden kann, sonst FALSE

Parameter *BcNr*: long

Clearingnummer der Bank. Falls als Type YellowBill (6) oder PayNet (7) übergeben wird, wird BcNr ignoriert.

Type: long

Art der Datei. Gültige Werte sind:

1 DebitDirect (senden)

2 BESRVESR (empfangen)

3 DTA (senden)

4 EZAG (senden)

5 MT940/MT950 bankauszüge (empfangen)

6 YellowBill (senden)

7 PayNet (senden)

2.11.4 EBanking.PoolDateienLoeschen

Boolean PoolDateienLoeschen *(Sequenznummer)*

Diese Funktion löscht alle Dateien aus dem Pool der pendenten EBanking- Aufträge mit der entsprechenden Sequenznummer.

Rückgabe TRUE, wenn Dateien gelöscht werden konnten, sonst FALSE

Parameter *Sequenznummer*: long

Sequenznummer der zu löschenden Pool-Dateien

2.11.5 EBanking.PoolnamenErstellen

String PoolnamenErstellen (DateiName, Type, SequenzNummer)

Generiert anhand des Original-Dateinames und dem Dateityp einen eindeutigen Namen für den Pool der pendenten Aufträge.

Rückgabe Dateinamen ohne Pfad.

Parameter *DateiName*: String

Originalnamen

Type: long

Art der Datei. Gültige Werte sind:

SequenzNummer: long

Nummer der Sequenz im PendenzenPool. Sie kann mit

NeueSequenzNummer() erstellt werden.

1 DebitDirect (senden)

2 BESRVESR (empfangen)

3 DTA (senden)

4 EZAG (senden)

5 MT940/MT950 Bankauszüge (empfangen)

6 YellowBill (senden)

7 PayNet (senden)

2.11.6 EBanking.SequenzNummerErstellen

Long SequenzNummerErstellen *(SequenyBasis)*

Generiert eine neue, eindeutige Sequenznummer. Diese Nummer wird verwendet, um Dateien im Pendenzenpool eindeutig zu identifizieren.

Rückgabe Sequenznummer

Parameter *SequenyBasis*: long

Muss 0 sein.

2.12 ESRParser

2.12.1 ESRParser Objekt

Bezeichnung Mit diesem Objekt können ESR Texte (vom Belegleser) und ESR Referenz-Nummern überprüft werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(22) erstellt.

Eigenschaften BankKto: String

Die Konto-Nr der Zahlungspflichtigen.

Betrag: Currency

Der Betrag, falls er auf dem ESR steht. Sonst ist der Betrag Fr. 0.00

Clearing: String

Die Clearing-Nr. Diese Nr wird normalerweise von REWE als BankId verwendet im Bank objekt.

ESRTyp: Integer

Der Typ des ESR. Der Wert entspricht eines der Werte, welche unter BankInfo.Zahlungsart im BankInfo Objekt gespeichert wird.

Index: String

Der Index, der verwendet wird um den betreffenden PK wieder zu finden.

Referenz: String

Die Referenz-Nr des ESR.

2.12.2 ESRParser.Parse

Integer Parse *(Text)*

Parst den ESR Text, der von einem Belegleser zurück gegeben wird und speichert die Werte in diesem Objekt. ESR Texte fangen mit einem "-" Zeichen an. Wenn die ESR Referenz-Nr eingegeben wird (ohne "-"), wird nur diese überprüft.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Text*: String

Der Text vom Belegleser, oder die Referenz-Nr.

Beispiel Das Beispiel verarbeitet einen Text vom Belegleser

```
Dim esr As Object
esr = man.NeuObjekt(22)
Dim esrText As String
esrText = "-0100000211459>101216864000004857071020048+010231059>"
If (esr.Parse(esrText) = 0) Then
MsgBox("Ref= " + esr.Referenz + " Index= " + esr.Index)
Else
MsgBox("ESR Lese-Fehler!")
End If
```

Siehe ESRParser, Referenz, Index

2.13 IBANParser

2.13.1 IBANParser Objekt

Bezeichnung Mit diesem Objekt können IBAN-Nummern überprüft und aufgeschlüsselt werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(23) erstellt.

Eigenschaften BankKto: String

Die Konto-Nr der Zahlungspflichtigen.

Clearing: String

Die Clearing-Nr. Diese Nr wird normalerweise von REWE als BankId verwendet im Bank objekt.

SWIFT: String

Liefert die SWIFT-Adresse der Bank. Dabei wird die Clearingnummer aus der IBAN verwendet. Falls keine Bank mit der entsprechenden Clearingnummer im Bankenstamm gefunden wurde oder diese keine SWIFT-Adresse gespeichert hat, wird ein Empty-String geliefert. im Bank objekt.

Check: Integer

Prüfziffer der IBAN-Nummer

Country: String

Landeskürzel der IBAN

2.13.2 IBANParser.Parse

Integer Parse *(Iban)*

Parst die IBAN und speichert die Werte in diesem Objekt.

Rückgabe 0 wenn erfolgreich, sonst einer der folgenden Fehler.

SFB_NullString

SFB_IbanDefIniNotFound

SFB_IbanUnknownCountryCode

SFB_IbanEmptyOrIllegalCharacter

SFB_IbanChecksumError

SFB_IbanLengthError

SFB_IbanUnknownCountryCode

Parameter *Iban*: String

Die IBAN-Nummer

2.14 Kontakt

2.14.1 Kontakt Objekt

Bezeichnung Mit diesem Objekt können Kontakte aus Adressen bearbeitet werden. Objekt wird mit der Funktion Mandant.NeuObjekt(24) erstellt

Eigenschaften Abteilung: String[31]

Die Abteilung.

AdressID: String[13]

Die Adresse zu der dieser Kontakt gehört.

Anrede: String[41]

Die Anrede.

Funktion: String[31]

Die Funktion.

GebDatum: Date

Das Geburtsdatum.

IstStandard: Boolean

Legt fest, ob dieser Kontakt als Standard verwendet werden soll.

KontaktNr: String[18]

Die Kontakt-Nr wird benützt, um einen Kontakt eindeutig zu identifizieren.

KontaktTyp: Integer

Folgende Kontakttypen sind definiert:

0 = Firmenkontakt (Adresse)

1 = Rechnungskontakt (Debitoren)

2 = Einkaufskontakt (Kreditoren)

Kuerzel: String[31]

Die Kuerzel.

Name: String[41]

Der Name.

Titel: String[41]

Der Titel.

Vorname: String[41]

Der Vorname.

Zustaendig: String[31]

Zustaendig

2.14.2 Kontakt.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Kontakt in die Datei. Die KontaktNr dieses Objekts darf noch nicht von einem anderen Eintrag verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Kontakt, Schreiben

2.14.3 Kontakt.Lesen

String[18] Lesen *(KontaktNr)*

Liest einen Kontakt von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *KontaktNr*: String[18]

Die Kontakt-Nr des Kontakts der gelesen werden soll.

Siehe Kontakt, KontaktNr

2.14.4 Kontakt.LesenRel

Integer LesenRel *(Wie)*

Liest einen Kontakt von der Datei relativ zum Eintrag, der sich jetzt im Kontakt-Objekt befindet. Wenn eine Adresse (AdressID) festgelegt wird, bevor diese Funktion aufgerufen wird, werden nur Kontakt vom der angegebenen Adresse zurückgegeben.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Kontakt gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei, resp der Adresse.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 oder 10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

3 oder 11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Beispiel Das C#-Beispiel fügt zwei Kontakte an die Adresse "10000" an und liest anschliessend alle Kontakte von der Adresse zeigt sie in einer Dialog Box an.

```
OMandant man = new OMandant ();
man.Login (2, @"C:\ProgramData\Sage\Data\Rewe\SageDemo16", null);

Kontakt kont = (Kontakt)man.NeuObjekt (24);

// Erster Kontakt einfuegen
kont.AdressID = "10000";
kont.KontaktNr = "10000.001";
kont.Vorname = "Hans";
kont.Name = "Meyer";
short res = kont.Einfuegen ();

// noch einen Kontakt
kont.KontaktNr = "10000.002";
kont.Vorname = "Walter";
kont.Name = "Abbegger";
res = kont.Einfuegen ();


// jetzt alle Kontakte lesen
kont.AdressID = "10000";
res = kont.LesenRel (0);
while (res == 0) {
MessageBox.Show (kont.Vorname + ' ' + kont.Name +  ',' +
kont.Abteilung);
res = kont.LesenRel (3);
}

// Sicherstellen, dass alle Ojekte vor
// Mandant-Objekt verschwinden.
kont = null;
GC.Collect ();
GC.WaitForPendingFinalizers ();
man.Logout ();
```

Siehe Kontakt, Lesen

2.14.5 Kontakt.Loeschen

Integer Loeschen()

Löscht den zuvor gelesenen Kontakt aus der Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Kontakt, Einfuegen

2.14.6 Kontakt.NextFreeNr

String[18] NextFreeNr *(AdressID)*

Gibt die zuletzt verwendete KontaktNr zurück

Rückgabe Die zuletzt verwendete KontaktNr + 1 oder AdressID + "000", wenn die Kontakt-Tabelle leer ist.

Parameter *AdressID*: String[13]

Aktuell AdressID

Siehe Kontakt

2.14.7 Kontakt.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Kontakt, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Kontakt, Einfuegen

2.14.8 Kontakt.SetIndex

Integer SetIndex *(Code)*

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code*: Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach KontaktNr.

1 : Sortiert nach AdressID & KontaktNr.

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

2.15 Konto

2.15.1 Konto Objekt

Bezeichnung Mit diesem Objekt können FibuNT Konten bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuKonto erstellt.

Eigenschaften AnzDaten: Long (nur Lesen)

Die Anzahl der Datensätze in der aktuellen Konten-Datei.

AnzHaben: Integer (nur Lesen)

Die Anzahl der Haben-Buchungen in diesem Konto.

AnzSoll: Integer (nur Lesen)

Die Anzahl der Soll-Buchungen in diesem Konto.

BetragHaben: Currency (nur Lesen)

Die Summe der Haben-Buchungen auf dieses Konto.

BetragHabenFW: Currency (nur Lesen)

Die Summe der Fremdwährungs Haben-Buchungen auf dieses Konto.

BetragSoll: Currency (nur Lesen)

Die Summe der Soll-Buchungen auf dieses Konto.

BetragSollFW: Currency (nur Lesen)

Die Summe der Fremdwährungs Soll-Buchungen auf dieses Konto.

Bezeichnung: String[31]

Die erste Zeile der Bezeichnung des Kontos.

Bezeichnung2: String[31]

Die zweite Zeile der Bezeichnung des Kontos.

BezeichnungFW: String[31]

Die erste Zeile der Bezeichnung des Kontos in der Sprache der Fremdwährung.

BezeichnungFW2: String[31]

Die zweite Zeile der Bezeichnung des Kontos in der Sprache der Fremdwährung.

Budget: Currency

Das Budget dieses Kontos.

Budget2: Currency

Das Budget dieses Kontos für das übernächste Jahr. Dieser Betrag wird zur Zeit in FibuNT nicht benützt, darf aber durchaus benützt werden.

Code: String[11]

Der frei definierbare Code des Kontos.

DatumCh: Date

Das Datum, an dem dieser Datensatzt zuletzt bearbeitet wurde.

FlagBebuchbar: Boolean

Wenn dieser Schalter den Wert 'True' hat, kann das Konto im Rechnungswesen nicht manuell bebucht werden.

FlagKeineKontoNr: Boolean

Wenn dieser Schalter den Wert 'True' hat, wird in den Fibuauswertungen keine Konto-Nr für dieses Konto geschrieben.

FlagKeineLeeren: Boolean

Wenn dieser Schalter den Wert 'True' hat, wird in den Fibuauswertungen dieses Konto nicht aufgeführt, wenn es Leer ist.

FlagKontoRaffen: Boolean

Wenn dieser Schalter den Wert 'True' hat, werden im Kontoauszug alle Buchungen im Monat auf eine einzige zusammengezogen.

FlagReadOnly: Boolean

Wenn dieser Schalter den Wert 'True' hat, kann das Konto im Rechnungswesen nicht bearbeitet werden.

FlagBuchungsWarnung: Boolean

Wenn dieser Schalter den Wert 'True' hat, wird beim direkten Bebuchen dieses Kontos eine Warnung ausgegeben.

FwID: String[5]

Der Kürzel einer Fremdwährung definiert dieses Konto als Fremdwährungskonto.

IstGeheim: Boolean

Wenn diese Eigenschaft gesetzt ist (True), dann zeigen FibuNT, DebiNT und KrediNT, während dem Buchen das Saldo dieses Kontos nicht an.

IstHaben: Boolean (nur Lesen)

Diese Eigenschaft ist 'True' wenn das Konto eines der folgenden Typen hat (Siehe Konto.Typ): Passiv, Ertrag oder Haben-Kostenstelle

IstKstZwingend: Boolean

Wenn diese Eigenschaft 'True' ist, muss beim Buchen eine Kostenstelle für dieses Konto angegeben werden.

IstLeer: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn das Konto keine Buchungen enthält und keine Beträge in Konto.Vortrag, Konto.Vorjahr und Konto.Budget enthalten sind.

KontoNr: String[13] (Index)

Die Konto-Nr wird benützt, um ein Konto eindeutig zu identifizieren.

Kostenstelle: String[13]

Die Kostenstelle, die für dieses Konto beim Buchen vorgeschlagen werden soll.

Saldo: Currency (nur Lesen)

Der Saldo auf diesem Konto.

SaldoFW: Currency (nur Lesen)

Der Saldo auf diesem Konto in der Fremdwährung. Dieser Wert ist nur gültig, wenn das Konto auch ein FW-Konto ist. Siehe Konto.FwID

SteuerID: String[5]

Der Kürzel eines Steuersatzes. Dieser Steuersatzt wird beim Buchen automatisch vorgeschlagen.

Typ: Integer

Der Typ des Kontos. Diese Eigenschaft kann nur geändert werden, wenn keine Buchungen auf dem Konto sind. Folgendes sind zulässige Typen:

0 - Aktiv-Konto

1 - Passiv-Konto

2 - Aufwand-Konto

3 - Ertrags-Konto

4 - Hilfskonto-Soll

5 - Hilfskonto-Haben

6 - Hilfskostenstelle-Soll

7 - Kostenträger-Haben

8 - Vorkostenstelle-Soll

9 - Projekt-Haben

10 - Hauptkostenstelle-Soll

Vorjahr: Currency

Der Vorjahressaldo des Kontos.

Achtung: Ändert diese Eigenschaft, müssen die Bücher neu saldiert werden.

Vortrag: Currency

Der Saldovortrag des Kontos.

VortragFW: Currency

Der Saldovortag des Kontos in der Fremdwährung. Dieses Eigenschaft wird nur für Fremdwährungskonten bearbeitet. Siehe Konto.FwID.

2.15.2 Konto.Browser2

Integer Browser2 (Konto-Nr, Filter)

Mit dieser Methode kann der Benutzer mittels eines Browsers ein Konto suchen und lesen.

Rückgabe 0 wenn ein Konto gelesen wurde, sonst ein Fehlerstatus.

Parameter *Konto-Nr*: String[13]

Auf dieses Konto wird im Browser gescrollt. Ist der Parameter ein leerer String, positioniert der Browser auf dem ersten Datensatz.

Filter: Integer (optional)

Eine Bedingung die verursacht, dass nur bestimmte Konten aufgeführt werden:

1 - Nur Kostenstellen aufführen.

2 - Nur Konten, das heisst keine Kostenstellen aufführen.

Siehe Lesen, LesenRel

2.15.3 Konto.Einfuegen

Integer Einfuegen()

Schreibt ein neues Konto in die FibuNT Kontodatei. Die KontoNr des Kontos muss eine neue Nummer sein. Sie darf in der Kontodatei nicht vorhanden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst ein Fehlerstatus.

2.15.4 Konto.Leeren

Void Leeren()

Diese Funktion setzt alle Eigenschaften des Konto-Objekts auf seine Standard-Werte.

Rückgabe keine

Siehe Konto

2.15.5 Konto.Lesen

Integer Lesen *(KontoNr)*

Liest ein Konto von der Datei.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus

Parameter *KontoNr*: String[13]

Die Konto-Nr des Kontos, das gelesen werden soll.

Siehe Konto, KontoNr, LesenRel

2.15.6 Konto.LesenRel

Integer LesenRel *(Wie)*

Liest ein Konto von der Datei relativ zum Konto, das sich jetzt im Konto- Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie das nächste Konto gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Konto, Lesen

2.15.7 Konto.Saldieren

Integer Saldieren (DatumVon, DatumBis, Kostenart)

Diese Methode errechnet den Saldo des aktuellen Kontos gemäss den Buchungen, die im eingegebenen Datumsbereich stattgefunden haben. Die Saldi werden nicht permanent ins Konto geschrieben, sondern befinden sich in diesem Objekt bis ein neues Konto gelesen wird. Folgende Eigenschaften des Kontos sind von dieser Funktion betroffen: AnzHaben, AnzSoll, BetragHaben, BetragHabenFW, BetragSoll, BetragSollFW, Saldo und SaldoFW.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Parameter *DatumVon*: Date

Das Anfangsdatum.

DatumBis: Date

Das Schlussdatum.

Kostenart: String[13] (optional)

Eine KontoNr einer Kostenart. Wenn dieser Parameter angegeben wird, werden nur Buchungen verwendet, die auf die entsprechende Kostenart verweisen (siehe Buchung.GKontoNr).

Siehe Lesen

2.15.8 Konto.Schreiben

Integer Schreiben()

Schreibt das zuvor gelesene Fibu-Konto in die Konto-Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Kommentar Diese Funktion sollte mit grosser Vorsicht verwendet werden. Aenderungen an einigen Eigenschaften können grosse Auswirkungen haben auf den Verlauf des Programms.

Siehe Konto, Einfuegen

2.16 Kreis

2.16.1 Kreis Objekt

Bezeichnung Mit diesem Objekt können Beleg-Nummernkreise bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuKreis erstellt.

Eigenschaften BelegNr: Long (nur Lesen)

Die zuletzt benützte Beleg-Nr des Nummernkreises.

KreisID: String[11] (nur Lesen)

Der eindeutige Kürzel des Nummernkreises.

KontoNr: String[13] (nur Lesen)

Das Konto mit dieser Kontonummer wird standardmässig als Gegenkonto beim Buchen vorgeschlagen.

2.16.2 Kreis.Lesen

Integer Lesen *(KreisID)*

Liest einen Nummernkreis von der Datei.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *KreisID*: String[11]

Der Kürzel des Nummernkreises, der gelesen werden soll. Wenn dieser Parameter Null oder ein leerer String ist, wird der Nummernkreis 'Allgemein' benützt.

Siehe Kreis, KreisID, LesenRel

2.16.3 Kreis.LesenRel

Integer LesenRel *(Wie)*

Liest von der Datei einen Nummernkreis relativ zum Kreis, der sich im Kreis-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Wie*: Integer

Die Methode zum Lesen des nächsten Kreises:

0 (FIRST): Liest den ersten Datensatz der Datei.

3 oder 11 (NEXT): Liest den nächsten Datensatz.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei.

Beispiel Das Beispiel zeigt wie in FibuNT die Nummernkreis Combo-Box mit den bestehenden Kreisen des Mandanten gefüllt wird:

```
' Kreis-Objekt erstellen
Dim Kreis As Object
Set Kreis = Man.NeuKreis
wCombo.Clear

' Kreis 'Allgemein' lesen
res% = Kreis.Lesen("")
If res% = 0 Then
defKreis$ = Kreis.KreisID
wCombo.AddItem defKreis$
End If

' Schleife durch alle Kreise
res% = Kreis.LesenRel(0)
While res% = 0
If Kreis.KreisID <> defKreis$ Then
wCombo.AddItem Kreis.KreisID
End If
res% = Kreis.LesenRel(3)
Wend
```

Siehe Lesen

2.17 Kurs

2.17.1 Kurs Objekt

Bezeichnung Mit diesem Objekt können FW-Tageskurse bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(15) erstellt.

Eigenschaften GueltigAb: Date

Das Datum ab dem dieser Tageskurs verwendt werden soll.

ID: String[5]

Der eindeutige Kürzel der Fremdwährung.

ISOCode: String[5]

Der ISO Code der Währung.

ValorenNr: String[13]

Die Valoren-Nr.

GeldKurs: Single

Der Geld-Kurs dieser Währung.

BriefKurs: Single

Der Brief-Kurs dieser Währung.

MittelKurs: Single (nur Lesen)

Der Mittel-Kurs dieser Währung.

Siehe Mandant

2.17.2 Kurs.DateiErstellen

Integer DateiErstellen()

Erstellt eine neue leere Tageskurs-Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Siehe Kurs, Lesen

2.17.3 Kurs.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Kurs in die Kursdatei. Der Kurs-ID und das GueltigAb-Datum zusammen müssen eindeutig sein. Sie dürfen in der Kursdatei nicht schon vorhanden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst ein Fehlerstatus.

Siehe Kurs, Schreiben

2.17.4 Kurs.Lesen

Integer Lesen *(ID, Datum)*

Liest einen Kurs von der Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *ID*: String

Der Kurs-ID des Kurses, der gelesen werden soll.

Datum: Date

Das GueltigAb-Datum des Kurses, der gelesen werden soll.

Siehe Kurs, ID, GueltigAb, LesenRel

2.17.5 Kurs.LesenRel

Integer LesenRel *(Wie)*

Liest einen Kurs von der Datei relativ zum Kurs, der sich jetzt im Kurs- Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Kurs gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Kurs, Lesen

2.17.6 Kurs.Loeschen

Integer Loeschen()

Löscht den zuvor gelesene Kurs aus der Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst ein Fehlerstatus.

2.17.7 Kurs.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesene Kurs in die Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Kurs, Einfuegen

2.18 MahnIndex

2.18.1 MahnIndex Objekt

Bezeichnung Mit diesem Objekt kann die aktuelle Mahnliste ausgelesen und bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(16) erstellt.

Eigenschaften Datum: Date (nur Lesen)

Das Datum an dem der OP fällig wurde.

FwlID: String[5] (nur Lesen)

Die Fremdwährung des Personenkontos oder leer falls das PK in Leitwährung geführt wird.

Kurzname: String[11] (nur Lesen)

Der Kurzname oder Suchbegriff (SortID).

OPNr: String[13] (Index)

Die OP-Nummer des OPs der gemahnt werden soll.

PKNr: Long (nur Lesen)

Die Personenkontonummer.

Siehe Mandant

2.18.2 MahnIndex.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Eintrag in die Mahnliste. Bevor diese Funktion aufgerufen wird muss das Feld OPNr gesetzt werden.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe MahnIndex, Loeschen

2.19 MahnIndex.Lesen

Integer Lesen *(Op-Nr)*

Liest einen Eintrag in der Mahnliste.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Op-Nr*: String[13]

Der OP-Nr eines Eintrags in der Mahnliste.

Siehe MahnIndex, LesenRel

2.19.1 MahnIndex.LesenRel

Integer LesenRel *(Wie)*

Liest einen Eintrag von der Datei relativ zum Eintrag, die sich jetzt im MahnIndex-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Datensatz gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 oder 10 (PREVIOUS): Liest den vorhergehenden Datensatz mit der Operation ReadPrevious.

3 oder 11 (NEXT): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei.

Siehe MahnIndex

2.19.2 MahnIndex.Loeschen

Integer Loeschen()

Löscht den aktuellen Datensatz in diesem Objekt. Bevor diese Funktion aufgerufen wird muss das Feld OPNr gesetzt werden.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Siehe MahnIndex, Einfuegen

2.20 Mandant

2.20.1 Mandant Objekt

Bezeichnung Dieses Objekt stellt einen Rechnungswesen-Mandant dar. Durch Methoden (Funktionen) von diesem Objekt können alle anderen Objekte erstellt werden. Durch das Erstellen eines Objektes von diesem Typ wird die SOK- Schnittstelle von REWE geladen.

Eigenschaften DatumAnfang: Date (nur Lesen)

Das Datum des ersten Tages dieses Mandanten.

DatumEnde: Date (nur Lesen)

Das Datum des letzten Tages dieses Mandanten. Streng genommen ist es der letzte Tag eines Kalenderjahres nach dem Anfangsdatum.

IsSQLMandate: Boolean (nur Lesen)

Diese Eigenschaft ist für den internen Gebrauch reserviert.

IstNurLesbar: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn der Mandant als 'Read only' Mandant geöffnet wurde. Der Wert ist nur gültig, wenn ein Mandant.Login gemacht wurde.

IstVorlage: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn der Mandant als 'Vorlagen' Mandant gespeichert wurde.

KrediMode: Boolean

Diese Eigenschaft wird normalerweise automatisch gesetzt gemäss dem entsprechenden Mandant.Login. Im KrediModus werden die OP und PK Objekte alloziert um mit Kreditoren (Lieferanten) zu arbeiten.

Name: String (nur Lesen)

Der Name des Mandanten (Siehe Mandant.IniLesen).

Passwort: String (nur Lesen)

Gibt das Passwort zurück, mit dem dieser Mandant geöffnet wurde. Der Mandant muss offen sein damit ein gültiges Passwort zurückgegeben wird.

Pfad: String (nur Lesen)

Das Verzeichnis in dem sich der Mandant befindet.

PfadVorjahr: String (nur Lesen)

Das Verzeichnis in dem sich der Vorjahres-Mandant befindet.

PfadFolgejahr: String (nur Lesen)

Das Verzeichnis in dem sich der Mandant des Folgejahres befindet.

Version: Integer (nur Lesen)

Liefert die Version des Mandanten als Integer-Wert. Die beiden letzten

Stellen sind die Low-Version. z.B. 550 bedeutet Version 5.50

InMemoryCacheEnabled: Boolean

Mit dieser Eigenschaft kann der In-Memory Cache für SQL Mandanten ein- oder ausgeschaltet werden. Für Details dazu, was der In-Memory Cache ist, lesen Sie bitte den Abschnitt „In-Memory Cache für SQL Mandanten“ weiter oben in diesem Dokument. Die Eigenschaft kann nur abgefragt oder gesetzt werden, wenn zuvor Mandant.Login() erfolgreich aufgerufen worden ist und es sich bei dem Mandanten um einen SQL Mandanten handelt. Hat noch kein Login stattgefunden, oder der Mandant ist ein Btrieve Mandant, so gibt die Eigenschaft beim Abfragen immer false zurück bzw. das Setzen der Eigenschaft hat keinen Effekt (der Eigenschaft-Wert verbleibt auf „false“, auch wenn man die Eigenschaft auf den Wert „true“ setzt). Per Default hat die Eigenschaft den Wert „false“, der In-Memory Cache ist also abgeschaltet.

Beispiel Ein neues Mandanten-Objekt erstellen:

```
Dim Mandant As Object
Set Mandant = CreateObject("FibuNT.Mandant")
```

2.20.2 Mandant.AnzBuchungen

Long AnzBuchungen()

Diese Funktion ermittelt die Anzahl der Buchungen in diesem Mandanten.

Rückgabe Anzahl der Buchungen oder -1 wenn die Anzahl nicht ermittelt werden kann.

Siehe Mandant

2.20.3 Mandant.AnzKonten

Long AnzKonten()

Diese Funktion ermittelt die Anzahl der Konten in diesem Mandanten.

Rückgabe Anzahl der Konten oder -1 wenn die Anzahl nicht ermittelt werden kann.

Siehe Mandant

2.20.4 Mandant.AnzOP

Long AnzOP()

Diese Funktion ermittelt die Anzahl der OPs in diesem Mandanten.

Rückgabe Anzahl der OPs oder -1 wenn die Anzahl nicht ermittelt werden kann.

Siehe Mandant

2.20.5 Mandant.AnzPK

Long AnzPK()

Diese Funktion ermittelt die Anzahl der PKs in diesem Mandanten.

Rückgabe Anzahl der PKs oder -1 wenn die Anzahl nicht ermittelt werden kann.

Siehe Mandant

2.20.6 Mandant.BackupToPath

Diese Funktion ist für den internen Gebrauch reserviert.

2.20.7 Mandant.BelegLoeschen

Integer BelegLoeschen *(BelegNr)*

Löscht einen Beleg des aktuellen Mandanten.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *BelegNr*: Long

Eine Beleg-Nummer.

Kommentar Vorsicht diese Funktion wir einen Beleg unwiederrufbar löschen.

Wichtig! Wenn ein Debitoren-Beleg gelöscht wird, muss der Mandant als Debi-Mandant geöffnet werden, und umgekehrt für Kreditoren-Belege.

Siehe NeuBeleg, Beleg, Login

2.20.8 Mandant. BelegVerbuchen

Integer BelegVerbuchen *(BelegNr, Op-Nr)*

Verbucht einen vorerfassten Beleg. Der Beleg wird vom vorerfassten Zustand zum definitiv erfassten Zustand übertragen. **Rückgabe** 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *BelegNr*: Long

Die Beleg-Nr des vorerfassten Belegs, der verbucht werden soll.

Op-Nr: String

Eine OP-Nummer des vorerfassten Belegs.

Siehe Mandant

2.20.9 Mandant.CheckIBAN

Integer CheckIBAN *(String)*

Prüft einen IBAN-Account

Rückgabe 0 wenn erfolgreich

1 Das IBAN-Definitionsfile (IBANDEF.ini) konnte nicht gefunden werden.

Gleicher Ort wie Datalib

2 Der Ländercode konnte nicht gefunden werden.

3 Ungültiges Zeichen gefunden.

4 Prüfzifferfehler

5 Länge der IBAN-Nummer stimmt nicht

Parameter *String*: IBAN

Zu prüfender IBAN-Account.

2.20.10 Mandant.Formatieren

String Formatieren *(Zahl)*

Diese Funktion formatiert Beträge (Currency) und andere Zahlen gemäss den Einstellungen in FibuNT (Trennzeichen, Kommazeichen, usw.).

Rückgabe Ein String mit dem formatierten Wert.

Parameter *Zahl*: Variant

Die Zahl, die gemäss FibuNT-Einstellungen formatiert werden soll.

2.20.11 Mandant.FreieBelegNr

Long FreieBelegNr *(BelegNr)*

Holt die nächste freie Belegnummer; diese wird nicht reserviert. Die Funktion Beleg.Einfuegen wird automatisch die nächste freie Belegnummer holen, wenn inzwischen eine andere Anwendung im Netzwerk Ihre Belegnummer verwendet hat.

Wichtig: Diese Funktion berücksichtigt keine Nummernkreise. Falls mit Nummernkreisen gearbeitet wird, ist folgendes Vorgehen erforderlich, um eine freie Belegnummer zu ermitteln. Zuerst mit Mandant.LetzteBlgNrLesen(Nummernkreis) die letzte „gespeicherte“ Belegnummer des angegebenen Nummernkreis lesen. Danach die Belegnummer manuell inkrementieren. Die inkrementierte Belegnummer entspricht dann der freien Belegnummer innerhalb des angegeben Nummernkreis. Danach ist zwingend erforderlich, dass nach erfolgreicher Beleg-Speicherung die inkrementierte Belegnummer via Mandant.LetzteBlgNrSchreiben(BelegNr, Nummernkreis) wieder zurück geschrieben wird.

Rückgabe Eine freie Belegnummer.

Parameter *BelegNr*: Long

Eine Belegnummer, ab der gesucht werden soll.

Siehe Beleg.SetBelegNr

2.20.12 Mandant.FreieOpNr

String FreieOpNr *(Op-Nr)*

Holt die nächste freie OP-Nummer.

Rückgabe Eine freie OP-Nummer oder einen leeren String, wenn ein Fehler aufgetreten ist.

Parameter *Op-Nr*: String

Eine OP-Nummer, ab der gesucht werden soll.

Siehe FreieBelegNr, FreiePkNr

2.20.13 Mandant.FreiePkNr

Long FreiePkNr *(Pk-Nr)*

Holt die nächste freie PK-Nummer.

Rückgabe Eine freie PK-Nummer.

Parameter *Pk-Nr*: Long

Ab dieser Pk-Nummer soll gesucht werden.

Siehe FreieBelegNr, FreieOpNr

2.20.14 Mandant.HasRight

Long HasRight (ID des Rechtes)

Prüft auf ein Recht

Rückgabe 0 Erfolg

Parameter *ID des Rechtes*: clsid

Zu prüfendes Recht

2.20.15 Mandant.IniLesen

String IniLesen *(Sektion, Key)*

Mit dieser Funktion können Stammdaten und Einstellungen von dem Mandant-Objekt gelesen werden.

Rückgabe Der gelesene Mandanten-Eintrag. Ist kein Eintrag in der Stammdaten Datei vorhanden, wird ein leerer String zurück gegeben.

Parameter *Sektion*: Integer

Die Sektionsnummer, in der sich die Info befindet.

Key: Integer

Der Key innerhalb der Sektion.

Kommentar Eine Liste der gültigen Sektionen und Keys finden Sie im Benutzerhandboch von Sage 50 Rechnungswesen (Optionen – Einstellungen – Sektionen).

Siehe IniSchreiben

2.20.16 Mandant.IniSchreiben

Bool IniSchreiben (Sektion, Key, Text)

Mit dieser Funktion können Stammdaten und Einstellungen von dem Mandant-Objekt geändert werden.

Rückgabe 'True' wenn erfolgreich, sonst 'False'.

Parameter *Sektion*: Integer

Die Sektionsnummer in der sich die Info befindet.

Key: Integer

Der Key innerhalb der Sektion.

Text: String

Den Text den Sie schreiben wollen.

Kommentar Diese Funktion sollte nur begrenzt eingesetzt werden. Für die meisten Einträge gibt es in den Sage50 Anwendungen benutzergeführte Dialoge, die vom Benutzer nur gültige Werte akzeptieren.

Siehe IniLesen

2.20.17 Mandant.IstOffen

Boolean IstOffen()

Mit dieser Funktion können Sie abfragen, ob das Mandant-Objekt schon offen ist, d.h. ob ein Login schon erfolgreich durchgeführt wurde.

Rückgabe TRUE wenn der Mandant offen ist, sonst FALSE

2.20.18 Mandant.LetzteBlgNrLesen

Long LetzteBlgNrLesen *(Nummernkreis)*

Diese Funktion gibt die zuletzt benützte FibuNT Belegnummer eines bestimmten Nummernkreises zurück.

Rückgabe Eine FibuNT Belegnummer.

Parameter *Nummernkreis*: String

Die ID eines Nummernkreises. Wenn dieser Parameter ein leerer String ist, wird der Nummernkreis 'Allgemein' benützt.

Siehe FreieBelegNr, LetzteBlgNrSchreiben

2.20.19 Mandant.LetzteBlgNrSchreiben

Boolean LetzteBlgNrSchreiben *(BelegNr, Nummernkreis)*

Mit dieser Funktion können Sie in einem der bestehenden Nummernkreise eine Belegnummer speichern, die zum Speichern eines Beleges zuletzt von Ihrer Anwendung benützt wurde. Das ermöglicht FibuNT für einen bestimmten Nummernkreis eine freie Belegnummer schneller zu ermitteln.

Rückgabe 'True' wenn erfolgreich, sonst 'False'.

Parameter *BelegNr*: Long

Die zuletzt benützte Belegnummer des Nummernkreises.

Nummernkreis: String

Der Id eines Nummernkreises. Wenn dieser Parameter ein leerer String ist, wird der Nummernkreis 'Allgemein' benützt.

Siehe FreieBelegNr, LetzteBlgNrLesen

2.20.20 Mandant.LetzterMandant

String (nur Lesen) LetzterMandant()

Gibt den Pfad des zuletzt verwendeten Mandanten zurück, d.H den Mandanten mit dem in FibuNT, DebiNT oder KrediNT zuletzt gearbeitet wurde.

2.20.21 Mandant.Login

Integer Login (Typ, Pfad, Passwort)

Öffnet einen Mandanten. Es können entweder die INI-Datei, die FibuNT- Dateien oder die DebiNT/KrediNT Dateien geöffnet werden.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Typ*: Integer

Der gewünschte Typ von Login.

0 - nur die INI-Datei

1 - die FibuNT-Dateien

2 - die DebiNT-Dateien

3 - die KrediNT-Dateien

Pfad: String (optional)

Der volle Pfad eines FibuNT, DebiNT oder KrediNT Mandanten, d.h. das

DOS-Verzeichnis des Mandanten.

Wenn dieser Parameter nicht angegeben wird, wird eine Dialog-Box präsentiert, damit der Benutzer einen Mandanten selbst wählen kann.

Passwort: String (optional)

Hat ein Mandant kein Passwort, so darf hier nichts angegeben werden. Hat der Mandant ein Passwort und es wird hier nichts angegeben, so erscheint eine interaktive Password-Eingabemaske. Hat der Mandant ein Passwort und es wird hier angegeben, so erscheint die Eingabemaske nicht. Wird ein falsches Passwort übergeben, so wird eine Fehlermeldung angezeigt und das Öffnen des Mandanten schlägt fehl. WICHTIG: Das Mandanten-Passwort unterstützt keine Gross-/Kleinschreibung. Das Passwort muss deshalb zwingend in Grossbuchstaben umgewandelt übergeben werden.

Beispiel Ein FibuNT-Mandant öffnen:

```
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")
res% = Man.Login(1, "C:\ProgramData\Sage\Data\Rewe\SageDemo16")
If res% <> 0 Then
MsgBox "Der Mandant wurde nicht geöffnet.", MB_OK
Exit Sub
End If
```

Siehe Logout

2.20.22 Mandant.Logout

Integer Logout()

Schliesst den offenen Mandanten.

WICHTIG: Vor diesem Funktionsaufruf sollten zuerst alle Objekte, die über das Mandanten Objekt erstellt wurden, zerstört werden.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Siehe Login

2.20.23 Mandant.NeuAdresse

Adresse NeuAdresse()

Siehe die Funktion NeuObjekt(2)

2.20.24 Mandant.NeuBeleg

Beleg NeuBeleg()

Siehe die Funktion NeuObjekt(3)

2.20.25 Mandant.NeuBuchung

Buchung NeuBuchung()

Siehe die Funktion NeuObjekt(4)

2.20.26 Mandant.NeuKonto

Konto NeuKonto()

Mit dieser Funktion erstellen Sie ein neues Konto-Objekt mit dem Sie Konten in der Fibu bearbeiten.

Rückgabe Konto oder Null, wenn nicht genug Speicher vorhanden ist.

Beispiel Ein neues Konto allozieren:

```
Dim Konto As Object

Set Konto = Mandant.NeuKonto()
```

Siehe Mandant, Konto

2.20.27 Mandant.NeuKreis

Kreis NeuKreis()

Siehe die Funktion NeuObjekt(6)

2.20.28 Mandant.NeuObjekt

Object NeuObjekt *(Typ)*

Mit dieser Funktion erstellen Sie ein neues Speicherobjekt mit dem Sie bestimmte Datensätze bearbeiten können. Zum Beispiel Mandant.NeuObjekt(1) erstellt ein Konto-Objekt, mit dem Sie FibuNT-Konten dieses Mandanten bearbeiten können. Das Konto-Objekt ist noch nicht mit einem bestimmten Konto verbunden, sondern es stellt nur einen temporären Speicherplatz dar. Sie müssen jetzt das gewünschte Konto in das Objekt einlesen, irgend welche Aenderungen machen, dann wieder zurückschreiben. Das machen Sie mit den Methoden und Eigenschaften des Konto-Objekts.

Rückgabe Das Objekt des angegebenen Typs oder Null, wenn nicht genug Speicher vorhanden ist, oder der Mandant nicht richtig offen ist (Login).

Parameter *Typ*: Integer

Der Typ des Objektes das erstellt werden soll. In dem Kommentar wird die Liste der Objekte aufgezählt.

Kommentar Nachfolgend ist eine Liste der verschiedenen Objekte. Um Objekte die mit * gekennzeichnet sind zu erstellen, muss dieser Mandant als Debi-/ oder Kredi-Mandant geöffnet worden sein.

1 **Konto**-Objekt um Konten in der Fibu zu bearbeiten.

2 **Adresse**-Objekt um Adressen der PKs zu bearbeiten.

3 **Beleg**-Objekt um Belege/Buchungen in der Fibu zu bearbeiten.

4 **Buchung**-Objekt um einzelne Buchungen in der Fibu zu bearbeiten.

5* **Zahlbed**-Objekt um Zahlungsbedingungen zu bearbeiten. Wenn die Zahlungsbedingungsdatei nicht geöffnet werden kann wird kein Objekt zurückgegeben.

6 **Kreis**-Objekt um Nummernkreise zu bearbeiten.

7* **OP**-Objekt um Offene-Posten zu bearbeiten.

8* **PK**-Objekt um P-Konten zu bearbeiten.

9 **Steuer**-Objekt um Steuersätze von FibuNT zu bearbeiten.

10 **Struktur**-Objekt um Struktureinträge von FibuNT zu bearbeiten.

11 **Waehrung**-Objekt um Fremdwährungen von FibuNT zu bearbeiten.

12 **Methode**-Objekt um externe KST/KTR Methoden zu bearbeiten.

13 **Bank**-Objekt um interne Banken zu bearbeiten.

14 **Bankstamm**-Objekt um Banken im Bankenstamm zu bearbeiten.

15 **FwKurs**-Objekt um Fremdwährungskurse zu bearbeiten.
Dieses ist ein internes Object, das sich noch ändern kann.

16 **MahnIndex**-Objekt um die Mahnliste zu lesen.

17 **Budget**-Objekt um Budget zu bearbeiten.

18 **BuchArchiv**-Objekt um Buchungen im Archiv zu lesen.

20 **EBanking**-Objekt um Einstellungen über das EBanking auszulesen. Falls der Mandant nicht für EBanking konfiguriert ist, liefert die Funktion NULL zurück.

21 **BankInfo**-Objekt um Bankverbindungen zu bearbeiten.

22 **ESRParser**-Objekt um ESR/Referenz zu parsen/verifizieren.

23 **IBANParser**-Objekt um IBAN zu parsen/verifizieren.

24 **Kontakt**-Objekt um Kontakte zu bearbeiten.

25 **Anschrift**-Objekt um Anschriften zu bearbeiten.

26 **PlzStamm**-Objekt um Daten im PLZ-Stamm zu bearbeiten.

27 **Dauerauftrag**-Objekt um Daueraufträge zu bearbeiten.

41 **OP**-Objekt fuer KNV!

42 **Buchung**-Objekt fuer KNV.

Beispiel Ein neues Konto-Objekt allozieren:

```
Dim Konto As Object
Set Konto = Mandant.NeuObjekt(1)
```

Siehe Mandant

2.20.29 Mandant.NeuOP

OP NeuOP()

Siehe die Funktion NeuObjekt(7)

2.20.30 Mandant.NeuPK

PK NeuPK()

Siehe die Funktion NeuObjekt(8)

2.20.31 Mandant.NeuSteuer

Steuer NeuSteuer()

Siehe die Funktion NeuObjekt(9)

2.20.32 Mandant.NeuStruktur

Struktur NeuStruktur()

Siehe die Funktion NeuObjekt(10)

2.20.33 Mandant.NeuWaehrung

Waehrung NeuWaehrung()

Siehe die Funktion NeuObjekt(11)

2.20.34 Mandant.NeuZahlbed

Zahlbed NeuZahlbed()

Siehe die Funktion NeuObjekt(5)

2.20.35 Mandant.PathContainsMandate

Diese Funktion ist für den internen Gebrauch reserviert.

2.20.36 Mandant.RebuildDatabase

Diese Funktion ist für den internen Gebrauch reserviert.

2.20.37 Mandant.SetSecurity

Long SetSecurity (Security-Manager, SSID, User-Name, User-Passwort, Computername)

Aktiviert sie Benutzer- un Rechteverwaltung Der Methode Mandant::Login muss kein Mandantenpasswort mehr mitgegeben werden!

Rückgabe 0 Erfolg

Parameter *Security-Manager*: Manager

Falls bereits eine Instanz eines Securitymanages existiert, kann diese verwendet werden. Die Parameter Pw und Host werden ignoriert, die Session und der User ist aber zwingend.

SSID: Session

Falls mit einer bestehenden Instanz eines Securitymanagers gearbeitet wird, muss die SSID mitgegeben werden.

User-Name: User

Der Username muss in jedem Fall angegeben werden!

User-Passwort: Pw

Falls ohne bestehenden Securitymanager gearbeitet wird, muss das Userpasswort angegeben werden.

Computername: Host

Falls ohne bestehenden Securitymanager gearbeitet wird, muss der Name des Computers angegeben werden.

2.21 Methode

2.21.1 Methode Objekt

Bezeichnung Mit diesem Objekt können KST/KTR Methoden bearbeitet werden. Falls dieser Mandant noch keine Methoden-Datei enthält, können sie mit der Funktion Methode.DateiErstellen eine neue Datei erstellen.

Eigenschaften Anteil: Double

Der Anteil-Wert.

Basis: Double

Der Basis-Wert für die Berechnung mit der Verteilerart 4.

KstId: String[13]

Der ID eines Kostenträgers.

MetId: String[13]

Der ID der Methode.

Prioritaet: Integer

Die Priorität der Verteilung des Restbetrags.

Verteilart: Integer

Die Verteilart, die festlegt wie der Methode.Anteil berechnet wird. Nur die folgenden Verteilarten sind zulässig:

1 - Anteil ist ein Prozent des Restbetrags

3 - Anteil ist ein Festbetrag.

4 - Anteil und Basis bilden einen Faktor.

99 - Der Restbetrag.

2.21.2 Methode.DateiErstellen

Integer DateiErstellen()

Erstellt eine neue Methoden-Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Kommentar Die Funktion gibt einen Fehler zurück wenn schon eine Methoden-Datei für diesen Mandanten besteht. Die Datei heisst 'sfbMeth.dat' und wird im Verzeichnis des aktuellen Mandanten erstellt.

Siehe Methode, Einfuegen

2.21.3 Methode.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Eintrag in die Methoden-Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Beispiel Das Beispiel schreibt einen Datensatz in die Methoden-Datei.

```
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")
res = Man.Login(1, "C:\ProgramData\Sage\Data\Rewe\SageDemo16")
If (res <> 0) Then
MsgBox "Login geht nicht."
Exit Sub
End If

Dim Met As Object
Set Met = Man.NeuObjekt(12)
Met.MetId = "9002"
Met.KstId = "9001"
Met.Anteil = 30
Met.Verteilart = 1
Met.Prioritaet = 1
res = Met.Einfuegen()
If (res <> 0) Then
MsgBox "Schreibfehler, schade!"
End If
```

Siehe Methode, Schreiben

2.21.4 Methode.LesenRel

Integer LesenRel *(Wie)*

Liest einen Methoden-Eintrag von der Datei relativ zum Eintrag, der sich jetzt im Methode-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Fremdwährung gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Beim Lesen mit Wie = 1 müssen folgende Eigenschaften gesetzt werden: MetId, Prioritaet und KstId. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe Methode, Lesen

2.21.5 Methode.Loeschen

Integer Loeschen()

Entfernt den zuvor gelesenen Eintrag aus der Methoden-Datei.

Rückgabe 0 wenn erfolgreich gelöscht, sonst einen Fehlerstatus.

Siehe Methode, LesenRel

2.21.6 Methode.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Eintrag in die Methoden-Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Methode, Einfuegen

2.22 OP

2.22.1 OP Objekt

Bezeichnung (Offener Posten) Mit diesem Objekt können DebiNT und KrediNT OPs bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuOP erstellt.

Eigenschaften AdrId: String[13]

Die Adressid

AnzHaben: Integer (nur Lesen)

Die Anzahl der Haben-Buchungen, die auf diesen OP gebucht wurden.

AnzSoll: Integer (nur Lesen)

Die Anzahl der Soll-Buchungen, die auf diesen OP gebucht wurden.

AnzZahlungen: Integer

Die Anzahl der Teilzahlungen, die schon gemacht wurden mit diesem OP.

BankId: String[13]

Identifikation der Zielbank

BankKto: String[25]

Das Bankkonto der Zielbank

BankNr: String[5]

Die BankNr einer internen Bank (Quellbank), mit der dieser OPs bezahlt werden sollen.

BetragFaktura: Currency

Der Rechnungs- bzw. Fakturabetrag des OPs.

BetragFakturaFW: Currency

Der Rechnungs- bzw. Fakturabetrag des OPs in der Fremdwährung.

BetragHaben: Currency (nur Lesen)

Die Summe der Haben-Buchungen, die auf diesen OP gebucht wurden.

BetragHabenFW: Currency (nur Lesen)

Die Summe der Fremdwährungs-Haben-Buchungen, die auf diesen OP gebucht wurden.

BetragSoll: Currency (nur Lesen)

Die Summe der Soll-Buchungen, die auf diesen OP gebucht wurden.

BetragSollFW: Currency (nur Lesen)

Die Summe der Fremdwährungs-Soll-Buchungen, die auf diesen OP gebucht wurden.

Code: String[11]

Der frei definierbare Code des OPs.

Datum: Date

Das Rechnungsdatum des OPs. (Gebrauchen bei Mahungen, etc.)

DatumCh: Date

Das Datum, an dem dieser Datensatzt zuletzt bearbeitet wurde.

Faellig: Date

Das Faelligkeitsdatum. Dieses Datum kann verwendet werden, um die Fälligkeit eines OP auf ein absolutes Datum zu setzen. Die Funktion OP.CalcFaellig wird dieses Datum nehmen, wenn dessen Wert nicht 0 ist.

FibuDatum: Date

Das FibuDatum des OPs (entspricht dem Datum der Entstehungsbuchung).

GFNr: Long

Die GFNr (Geschäftsfall-Nr) der Entstehungsbuchung dieses OPs.

IBAN: String[35]

Der IBAN-Account des OPs.

IstFW: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn das P-Konto (siehe OP.PKNr), in diesem OP ein Fremdwährungs-Konto ist.

IstKredi: Boolean (nur Lesen)

Diese Property hat den Wert 'True', wenn es sich um einen Kredi-OP handelt (Einstellung Haben).

Kontakt: String[40]

Der Kontakt-Name mit optionalen Ergänzungen.

LzBetrag: Currency

Der zuletzt bezahlte Betrag

LzSkonto: Currency

Das zuletzt verwendete Skonto

MahnDatum: Date

Das Datum der letzten Mahnung.

MahnStufe: Integer

Die Mahn-Stufe des OPs.

MahnverfahrenID: Integer

Das Mahnverfahren des OPs

Mitarbeiter: String[13]

Der Mitarbeiter für die OPs. Wird primär von der Vorerfassung verwendet.

OPNr: String[13] (Index)

Die OP-Nr wird benützt, um einen OP eindeutig zu identifizieren.

OpTyp: Integer

Der Type des OPs. Folgenden Typen sind definiert:

0 - Normaler OP.

1 - Akonto-/Vorauszahlungs OP.

Rechnungsart: Integer

Mögliche Werte sind:

0 = manuell (Papier)

1 = PayNet

2 = YellowBill

PKNr: Long

Die PK-Nr des P-Kontos mit der dieser OP verbunden ist.

Referenz: String[27] (Index)

Die Referenz-Nr des OPs. Diese Nummer ist normalerweise die ESR-Nr oder ein Index für den DTA Zahlungsverkehr.

ReferenzEndToEnd: String[36]

Eine Referenz, mit der der Offene Posten eindeutig auch ausserhalb von Sage 50 Rechnungswesen identifiziert werden kann. Die Referenz kann für beliebige Zwecke verwendet werden, sie wird aber insbesondere als End-To-End ID beim ISO 20022 Zahlungsverkehr eingesetzt (mit dem Ziel, dass beim Einlesen einer CAMT-Meldung eine Kreditoren-Zahlung, die sich in dem Kontoauszug in der CAMT-Meldung befindet, wieder dem Offenen Posten zugeordnet werden kann). Die Referenz ist zur Zeit als UUID implementiert, um die Einzigartigkeit zu gewährleisten. Man muss die Referenz aber immer als String behandeln und darf sich zudem nicht auf die UUID Formatierung verlassen - es ist denkbar, dass in Zukunft ein anderes Format verwendet wird, oder die Referenz gar keine UUID mehr ist.

Beim Aufruf von OP.Einfuegen() wird automatisch eine neue Referenz generiert. Beim Erstellen eines neuen Offenen Postens kann man die endgültige Referenz also erst **nach** dem Aufruf von OP.Einfuegen() in Erfahrung bringen.

ReferenzEndToEndOhneFormat: String[35]

Der Wert des Properties ReferenzEndToEnd ohne jegliche Formatierung.

Saldo: Currency (nur Lesen)

Der Saldo (Offener Betrag) auf diesem OP. Wenn der Saldo negativ ist, dann stellt der OP eine Gutschrift dar.

SaldoFW: Currency (nur Lesen)

Der Saldo (Offener Betrag) auf diesem OP in der Fremdwährung.

Dieser Wert ist nur gültig, wenn der OP ein FW-OP ist. Siehe OP.IstFW

Status: Short (nur Lesen)

Der Zahlungs-Status des OPs. Folgende Werte sind möglich:

0 - OP noch offen

1 - OP zur Zahlung vorgeschlagen

2 - OP bezahlt und abgeschlossen.

SteuerID: String[5]

Der Kürzel eines Steuersatzes, mit dem dieser OP verbunden ist.

Text: String[31]

Die Buchungstext des OP.

Vortrag: Currency

Der Saldovortrag des OPs.

VortragFW: Currency

Der Saldovortag des OPs in der Fremdwährung. Dieser Wert ist nur gültig, wenn das OP ein FW-OP ist. Siehe OP.IstFW.

Vorerfassen: Boolean

Handelt es sich um eine Vorerfassung?

Wichtig: Um den „vorerfassten“ OP vollständig und korrekt hinzuzufügen, ist erforderlich, dass nach dem Hinzufügen des OP via OP.Einfügen zusätzlich ein Beleg via Beleg Objekt erstellt wird. Dem Beleg Objekt muss dann via Property Beleg.Vorerfassen = True, ebenfalls mitgeteilt werden, dass es sich dabei um einen vorerfassten Beleg handelt.

Wird das Property auf dem Beleg nicht gesetzt, wird ein definitiver Beleg eingebucht. Im Sinne der Kreditoren-Vorerfassung liegt dann ein vorerfasster Kreditoren-OP mit einem definitiv eingebuchten Beleg vor, was einer inkonsistenten Datenhaltung entspricht. In solchen Fällen muss der vorerfasste Beleg sowie der „irrtümlich“ definitiv eingebuchte Beleg manuell gelöscht werden um die Inkonsistenz zu bereinigen.

ZahlDatum: Date

Das Datum der letzten Zahlung

Zahlungsart: int

Bezeichnet die Zahlungsart dieses OPs. Für Debitoren-Rechnungen sind die Zahlungsarten wie folgt:

0 - Manuell

1 - LSV

2 - IBAN/IPI

3 – LSV+

Für Kreditoren-Rechnungen sind folgende Codes definiert:

0 - Manuell

1 - ESR 15-stellig

2 - ESR 16-stellig

3 - ESR 27-stellig

4 - "Roter" (Bank)

5 - "Roter" (Post)

6 - "Roter" (Treuhand)

7 - Postmandat

8 - Fremdwährung

9 - IBAN/IPI

ZahlungslaufNr: int

Die Nummer des Zahlungslaufes, in dem der OP ist, oder 0.

ZblD: String[5]

Die Zahlungsbedingung des OPs.

Zahlungsgrund_1: String[30]

Der 1. Zahlungsgrund des OPs.

Zahlungsgrund_2: String[30]

Der 2. Zahlungsgrund des OPs.

Zahlungsgrund_3: String[30]

Der 3. Zahlungsgrund des OPs.

Zahlungsgrund_4: String[30]

Der 4. Zahlungsgrund des OPs.

2.22.2 OP.Browser

Integer Browser *(OPNr)*

Mit dieser Methode kann der Benutzer mittels Browsers ein OP suchen und lesen.

Rückgabe 0 wenn ein OP gelesen wurde, sonst einen Fehlerstatus.

Parameter *OPNr*: String[13]

Auf diesen OP wird im Browser gescrollt, wenn das Fenster sich öffnet. So steht der Benutzer gleich in der Nähe des OPs der ihn interessiert.

Beispiel Das Beispiel erstellt ein OP-Objekt und öffnet den Browser mit OP '950000' als Vorschlag:

```
Dim op As Object
Set op = Man.NeuOP()
res = op.Browser("950000") If (res <> 0) Then
MsgBox "Der Benutzer wollte keinen OP wählen."
Exit Sub
End If
```

Siehe OP, Lesen, LesenRel

2.22.3 OP.CalcFaellig

Date CalcFaellig()

Diese Funktion berechnet das Nettofälligkeitsdatum des OP.

Siehe OP, Faellig

2.22.4 OP.Einfuegen

Integer Einfuegen()

Schreibt einen neuen OP in die OP-Datei. Die OP-Nr. des OPs muss eine neue Nummer sein. Sie darf in der OP-Datei nicht vorhanden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Kommentar Um einen neuen OP zu erstellen haben Sie zwei Möglichkeiten. Sie benützen die Funktion Initialisieren, oder Sie lesen zuerst einen bestehenden, gültigen OP. Danach setzen Sie, für beide Methoden die OP- Nr auf einen neuen Wert, der noch nicht benutzt wurde. Beim Aufruf von Einfuegen() wird automatisch eine neue End-To-End Referenz generiert (Property ReferenzEndToEnd). Man kann die endgültige End-To-End Referenz also erst **nach** dem Aufruf von Einfuegen() in Erfahrung bringen.

Beispiel Folgendes Beispiel öffnet einen Mandant und erstellt für das Personenkonto mit Nr. 2 einen neuen OP, auf den gebucht werden kann:

```
Sub Command1_Click()
' Funktion um einen neuen OP zu erstellen
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")

' Den DebiNT Mandant öffnen
ans% = Man.Login(2, "C:\ProgramData\Sage\Data\Rewe\SageDemo16")
If (ans% <> 0) Then
MsgBox "Mandant konnte nicht geöffnet werden"
Exit Sub
End If

' OP-Objekt erstellen und initialisieren mit PK-Nr 2
Dim OP As Object Set
OP = Man.NeuOP
OP.Initialisieren (2)

' Nächste freie OP-Nr ab 10000
OP.OpNr = Man.FreieOpNr("10000")

' Neuer OP in die OP-Datei einfügen
ans% = OP.Einfuegen()
If (ans% = 0) Then
MsgBox "Yuhu! OP wurde erfolgreich geschieben."
End If
End Sub
```

Siehe OP, Mandant.FreieOpNr, Loeschen

2.22.5 OP.Initialisieren

Integer Initialisieren *(PK-Nr)*

Diese Funktion initialisiert einige Eigenschaften des OPs gemäss der PK- Nr. Den Rest der Eigenschaften werden auf Standard-Werte gesetzt.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus. Der Fehlerstatus bezieht sich auf einen Lesefehler des PKs.

Parameter *PK-Nr:* Long

Die PK-Nr eines PKs in dem Mandant von dem dieses Objekt stammt.

Kommentar Ein Beleg ist leer, wenn er mit der Funktion Mandant.NeuBeleg frisch erstellt wurde.

Siehe OP, Leeren

2.22.6 OP.Leeren

Void Leeren()

Diese Funktion setzt alle Eigenschaften des OPs auf seine Standard- Werte.

Rückgabe keine

Siehe OP, Initialisieren

2.22.7 OP.Lesen

Integer Lesen *(OPNr)*

Liest einen OP von der Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *OPNr:* String[13]

Die OP-Nr des OPs, der gelesen werden soll.

Siehe OP, OPNr, LesenRel

2.22.8 OP.LesenRel

Integer LesenRel *(Wie)*

Liest einen OP von der Datei relativ zum OP, der sich jetzt im OP-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste OP gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe OP, Lesen, SetIndex

2.22.9 OP.Loeschen

Integer Loeschen *(Man)*

Löscht den aktuellen Datensatz in diesem Objekt. Der OP muss zuerst gelesen werden bevor er gelöscht werden kann.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Man:* Mandant

Aktuellen Mandant

Kommentar Der OP kann nur gelöscht werden, wenn er ausgeglichen ist.

Siehe OP, Lesen, Einfuegen

OP, Beleg.Einfuegen

2.22.10 OP.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen OP in die OP-Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Kommentar Diese Funktion sollte mit grosser Vorsicht verwendet werden. Aenderungen an einigen Eigenschaften können grosse Auswirkungen haben auf den Verlauf des Programms.

Beispiel Das Beispiel erstellt ein OP-Objekt, liest den OP 950000 von der Datei, ändert das Fälligkeitsdatum auf das heutige Datum plus 30 Tage. Danach wird der OP wieder in die OP-Datei zurückgeschrieben. Zum Schluss wird noch angezeigt wann der OP demzufolge Fällig ist.

```
Dim op As Object
Set op = Man.NeuOP()
res = op.Lesen("950000")
If (res <> 0) Then
MsgBox "OP: 950000 ist nicht vorhanden."
Exit Sub
End If

op.Faellig = Now() + 30
res = op.Schreiben()
If (res <> 0) Then
MsgBox "OP: konnte nicht geschrieben werden.
Status = " + Str$(res)
End If
MsgBox "OP muss bis " + Str$(op.CalcFaellig()) + " bezahlt werden."
```

Siehe OP, Einfuegen

2.22.11 OP.SetIndex

Integer SetIndex *(Code)*

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code:* Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach OP-Nr.

1 : Sortiert nach OP-Referenz.

2 : Sortiert nach PK-Nr/Datum.

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

2.22.12 OP.ShowDocument

void ShowDocument *(DokumentId)*

Startet das Document Management System (DMS) und zeigt das Dokument das mit diesem OP verbunden ist.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *DokumentId:* String

Der ID des Dokumentes

Siehe OP

2.23 PK

2.23.1 PK Objekt

Bezeichnung (Personen-Konten) Mit diesem Objekt können P-Konten bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuPK erstellt.

Eigenschaften

AdressID: String[13] (nur Lesen) Die Adresse dieses P-Kontos.

Um die Adresse des PKs zu ersetzen benützen Sie die Funktion PK.SetAdressID.

AnzHaben: Integer (nur Lesen)

Die Anzahl der Haben-Buchungen, die auf dieses P-Konto gebucht wurden.

AnzOP: Integer (nur Lesen)

Die Anzahl der OPs von diesem P-Konto.

AnzSoll: Integer (nur Lesen)

Die Anzahl der Soll-Buchungen, die auf dieses P-Konto gebucht wurden.

BankId: String[13]

Identifikation der Zielbank

BankKto: String[25]

Das Bankkonto der Zielbank

BankNr: String[5]

Die BankNr einer internen Bank (Quellbank), mit der OPs von diesem PK bezahlt werden sollen.

BankVerbindung: String[13]

Bezeichnung der Standard-Bankverbindung

BereinigtPer: Date

Bis zu diesem Datum ist der Zahlungsverkehr bereinigt.

BetragHaben: Currency (nur Lesen)

Die Summe der Haben-Buchungen, die auf dieses P-Konto gebucht wurden.

BetragHabenFW: Currency (nur Lesen)

Die Summe der Fremdwährungs-Haben-Buchungen, die auf dieses P- Konto gebucht wurden.

BetragSoll: Currency (nur Lesen)

Die Summe der Soll-Buchungen, die auf dieses P-Konto gebucht wurden.

BetragSollFW: Currency (nur Lesen)

Die Summe der Fremdwährungs-Soll-Buchungen, die auf dieses P- Konto gebucht wurden.

Bezeichnung: String[31]

Die Bezeichnung des P-Kontos.

Budget: Currency

Der Budgetbetrag dieses PKs.

Code: String[9]

Der frei definierbare Code des P-Kontos.

DatumCh: Date

Das Datum, an dem dieser Datensatz zuletzt bearbeitet wurde.

DruckInfo: int

Die Adress-DruckInfo

EbppId: String[21]

Die EBPP-Teilnehmernummer wird verwendet, wenn ein Kunde Rechnungen auf elektonischem Weg erhalten möchte (EBPP).

EgIdent: String[15]

Stellt Bezeichner der EG-konformen MWST bereit

GKontoNr: String[13]

Ein Fibu-Konto, das als Gegen-Konto für dieses P-Konto vorgeschlagen wird.

IBAN: String[35]

Der IBAN-Account des PK.

IstEinmalKunde: Boolean

Legt fest, ob dieser PK ein Einmalkunde/Einmallieferant ist.

IstFW: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn das Konto (siehe PK.KontoNr), in diesem P-Konto auch ein Fremdwährungs-Konto ist. Diese Eigenschaft wird automatisch nachgeführt, wenn die Konto-Nr geändert wird.

IstKredi: Boolean (nur Lesen)

Diese Property hat den Wert 'True', wenn es sich um einen Kredi-PK handelt (Einstellung Haben).

KontaktPerson: String[31]

Die zuständige Kontakperson für dieses PK.

KontaktTelefon: String[21]

Die Telefon-Nr der zuständigen Kontaktperson für dieses PK.

KontoNr: String[13] (nur Lesen)

Das Fibu-Konto das mit diesem P-Konto verbunden ist. Normalerweise ist das ein Debitoren-, oder Kreditoren-Sammelkonto. Um das Konto zu ersetzen benützen Sie die Funktion PK.SetKontoNr.

Kreditlimite: Currency

Die Kreditlimite dieses PKs.

KundenNr: String[25]

Unsere Kunden-Nr bei diesem Kreditor (P-Konto).

Kurzname: String[11] (nur Lesen) Der Kurzname des PKs.

Diese Eigenschaft wird automatisch nachgeführt, wenn die Adresse des PKs geändert wird.

Mahngebuehrlock: Integer

Der Mahnsperr-Code. Wenn diese Eigenschaft auf 1 gesetzt wird, werden OPs von diesem PK nicht gemahnt.

Mahnlock: Integer

Der Mahnsperr-Code. Wenn diese Eigenschaft auf 1 gesetzt wird, werden OPs von diesem PK nicht gemahnt.

MahnverfahrenID: Integer

Das Mahnverfahren des PKs

PkIndex: String[17]

Index für OCR, ESR-Nummer

PKNr: Long (Index)

Die PK-Nr wird benützt, um ein P-Konto eindeutig zu identifizieren.

Saldo: Currency (nur Lesen)

Der Saldo auf diesem P-Konto.

SaldoFW: Currency (nur Lesen)

Der Saldo auf diesem P-Konto in der Fremdwährung. Dieser Wert ist nur gültig, wenn das P-Konto auch ein FW-Konto ist. Siehe PK.IstFW

Umsatz: Currency

Der Umsatz auf diesem P-Konto.

Umsatz1: Currency

Der Umsatz des Vorjahres auf diesem P-Konto.

Umsatz2: Currency

Der Umsatz des Vor-Vorjahres auf diesem P-Konto.

VKontoNr: String[13]

Ein Fibu-Konto, das als Gegen-Konto für dieses P-Konto vorgeschlagen wird, wenn eine Akonto-Rechnung gebucht wird.

Rechnungsart: int

Mögliche werte sind:

0 = manuell (Papier)

1 = PayNet

2 = YellowBill

Vortrag: Currency

Der Saldovortrag des P-Kontos.

VortragFW: Currency

Der Saldovortag des P-Kontos in der Fremdwährung. Dieser Wert ist nur gültig, wenn das P-Konto ein FW-PK ist. Siehe PK.IstFW.

Zahlungsart: Integer

Bezeichnet den Default der Zahlungsart dieses PKs. Für Debitoren sind die Zahlungsarten wie folgt:

0 - Manuell

1 - LSV

2 - IBAN/IPI

3 – LSV+

Für Kreditoren sind folgende Codes definiert:

0 - Manuell

1 - ESR 15-stellig

2 - ESR 16-stellig

3 - ESR 27-stellig

4 - "Roter" (Bank)

5 - "Roter" (Post)

6 - "Roter" (Treuhand)

7 - Postmandat

8 - Fremdwährung

9 - IBAN/IPI

Zahlungsbedingung: String[5]

Bezeichnet die Default-Zahlungsbedingung

2.23.2 PK.Browser

Integer Browser *(PK-Nr)*

Mit dieser Methode kann der Benutzer mittels Browsers ein PKonto suchen und lesen.

Rückgabe 0 wenn ein PK gelesen wurde, sonst einen Fehlerstatus.

Parameter *PK-Nr*: Long

Auf dieses PK wird im Browser gescrollt.

Siehe PK, Lesen, LesenRel

2.23.3 PK.Einfuegen

Integer Einfuegen()

Schreibt ein neues P-Konto in die PK-Datei. Die PKNr des P-Kontos muss eine neue Nummer sein. Sie darf in der PK-Datei nicht vorhanden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Beispiel Das Beispiel erstellt ein PK-Objekt, initialisiert es und fügt es in die PK-Datei ein:

```
Dim pk As Object
Set pk = Man.NeuPK()
pk.PKNr = 333
res% = pk.SetAdressID("00015")
res% = pk.SetKontoNr("1000")
res% = pk.Einfuegen()
```

Siehe PK

2.23.4 PK.Leeren

Void Leeren()

Diese Funktion setzt alle Eigenschaften des PKs auf seine Standard- Werte.

Rückgabe keine

Kommentar Ein Objekt das frisch von Mandant.NeuObjekt stammt ist schon geleert.

Siehe PK

2.23.5 PK.Lesen

Integer Lesen *(PKNr)*

Liest ein P-Konto von der Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *PKNr*: Long

Die PK-Nr des P-Kontos, das gelesen werden soll.

Siehe PK, PKNr, LesenRel

2.23.6 PK.LesenRel

Integer LesenRel *(Wie)*

Liest ein P-Konto von der Datei relativ zum P-Konto, das sich jetzt im PK- Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie das nächste P-Konto gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Siehe PK, Lesen

2.23.7 PK.Loeschen

Integer Loeschen()

Löscht den aktuellen Datensatz in diesem Objekt. Das PK muss zuerst gelesen werden bevor es gelöscht werden kann.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Kommentar Das PK kann nur gelöscht werden, wenn keine Buchungen darauf verweisen, d.H. AnzSoll und AnzHaben müssen beide 0 sein.

2.23.8 PK.Schreiben

Integer Schreiben()

Schreibt das zuvor gelesene P-Konto in die PK-Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Kommentar Diese Funktion sollte mit grosser Vorsicht verwendet werden. Aenderungen an einigen Eigenschaften können grosse Auswirkungen haben auf den Verlauf des Programms.

Beispiel Das Beispiel erstellt ein PK-Objekt, liest das PK 1 von der Datei, ändert zwei Eigenschaften und schreibt die Aenderungen wieder zurück in die PK- Datei:

```
Dim pk As Object
Set pk = Man.NeuPK()
res = pk.Lesen(1)
If (res <> 0) Then
MsgBox "PK: 1 ist nicht vorhanden."
Exit Sub
End If

pk.KontaktPerson = "Herr Meyer"
pk.KontaktTelefon = "(0800) 234 56 78"
res = pk.Schreiben()
If (res <> 0) Then
MsgBox "PK: konnte nicht geschrieben werden. Status = " +
Str$(res)
End If
```

Siehe PK, Einfuegen

2.23.9 PK.SetAdressID

Integer SetAdressID *(AdressID)*

Diese Methode ersetzt den Verweis der Adresse auf die dieses P-Konto. Die Adresse muss schon vorhanden sein. Die Eigenschaft Kurzname wird immer nachgeführt, und wenn die Eingenschaften Bezeichnung, KontaktPerson und KontaktTelefon leer sind, werden sie ebenfalls von der Adresse in das P-Konto kopiert.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *AdressID*: String[13]

Der ID einer gültigen Adresse.

Siehe PK, Adresse, AdressID

2.23.10 PK.SetIndex

Integer SetIndex (Code)

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code*: Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach PK-Nr.

1 : Sortiert nach PK-Index.

2 : Sortiert nach Adresse.

3 : Sortiert nach Kurzname

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

2.23.11 PK.SetKontoNr

Integer SetKontoNr *(KontoNr)*

Diese Methode ersetzt die Konto-Nummer auf die dieses PKonto verweist. Das Konto muss in der Fibu schon vorhanden sein. Die Eigenschaft IstFW wird auch nachgeführt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *KontoNr*: String[13]

Die Konto-Nr eines Fibu-Kontos.

Kommentar Anders als in FibuNT wird hier nicht geprüft ob das Konto den richtigen Typ hat. z.B. für Kreditoren sollten nur PASSIV-Konten verwendet werden.

Siehe PK, Konto, KontoNr

2.24 PlzStamm

2.24.1 PlzStamm Objekt

Bezeichnung Mit diesem Objekt können die Daten aus dem PLZ-Stamm bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuObjekt(26) erstellt.

Eigenschaften Kanton: String[2]

Der Kanton in der sich die Poststelle befindet.

Ort: String[25]

Der Ort in der sich die Poststelle befindet.

PlzNr: Long (Index)

Die PLZ als Nummer. Die PlzNr ist nicht eindeutig.

Siehe Mandant

2.24.2 PlzStamm.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Datensatz in die PLZ-Stamm Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe PlzStamm, Schreiben

2.24.3 PlzStamm.Lesen

Integer Lesen *(PlzNr)*

Liest einen Datensatz von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *PlzNr*: Long

Die PLZ die gelesen werden soll.

Kommentar Da die PlzNr nicht eindeutig ist kann diese Funktion höchstens dafür verwendet werden, um zu testen ob es mindestens einen Datensatz gibt mit der jeweiligen PLZ. In der Methode LesenRel ist ein Beispiel wie alle Orte zu einer PLZ gelesen werden.

Siehe PlzStamm, PlzNr

2.24.4 PlzStamm.LesenRel

Integer LesenRel *(Wie)*

Liest einen Datensatz von der Datei relativ zum Eintrag, die sich jetzt im PlzStamm-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Bankeintrag gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Beispiel Das C#-Beispiel liest und alle Ortschaften mit der PLZ 1000 und zeigt sie in einer Dialog Box an.

```
OMandant man = new OMandant ();
man.Login (2, @"C:\ProgramData\Sage\Data\Rewe\SageDemo16", null);

// Alle Orte mit PLZ 1000 (Lausanne) lesen
PlzStamm plz = (PlzStamm)man.NeuObjekt (26);
plz.PlzNr = 1000;
while (plz.LesenRel (3) == 0 && plz.PlzNr == 1000)
MessageBox.Show (plz.PlzNr.ToString () + ' ' + plz.Ort + ", "
+ plz.Kanton);

// Sicherstellen, dass alle Ojekte vor
// Mandant-Objekt verschwinden.
plz = null;
```

```
GC.Collect ();
GC.WaitForPendingFinalizers ();
man.Logout ();
```

Siehe PlzStamm, Lesen

2.24.5 PlzStamm.Loeschen

Integer Loeschen()

Löscht den zuvor gelesenen Datensatz aus der Datenbank.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe PlzStamm, Einfuegen

2.24.6 PlzStamm.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Dateneintrag, nach dem Aendern in die Datei zurück.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe PlzStamm, Einfuegen

2.24.7 PlzStamm.SetIndex

Integer SetIndex *(Code)*

Setzt den Zugriffs-Index auf einen anderen Sortierschlüssel. Mit der Funktion LesenRel wird dieser Index verwendet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Code*: Integer

Folgende Sortierschlüssel sind möglich:

0 : Sortiert nach PLZ, Ort, Kanton.

1 : Sortiert nach Ort, Kanton, PLZ.

Kommentar Wenn diese Methode nicht verwendet wird, wird automatisch Code 0 verwendet.

2.25 Steuer

2.25.1 Steuer Objekt

Bezeichnung Mit diesem Objekt können FibuNT Steuersätze bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuSteuer erstellt.

Eigenschaften BuchText: String[19]

Das Format für den automatisch generierten Buchungstext.

IstEingabeNetto: Boolean

Diese Eigenschaft legt fest, ob beim Buchen, Netto- oder Bruttobeträge eingegeben werden.

Konto: String[13]

Auf dieses Fibu-Konto soll der Steuerbetrag gebucht werden. Achten Sie darauf, dass das Fibu-Konto gültig ist.

KontoEw: String[13]

Ein weiteres Konto auf das Erwerbesteuer gebucht werden kann. Wird zur Zeit in FibuNT nicht benützt.

Prozent: Single

Der anzuwendente Prozentsatz (Siehe Steuer.BetragRechnen).

Quote: Single

Ein Quotient, durch den der Brutto-Betrag geteilt wird, bevor der Steuer.Prozent-Satz angewendet wird (Siehe Steuer.BetragRechnen).

SaldoSatz: Single

Der interne Abrechnungssatz für die MWST-Abrechnung, wenn nach Pauschal/Saldosätzen abgerechnet wird.

SteuerID: String[5]

Der eindeutige Kürzel des Steuersatzes.

2.25.2 Steuer.BetragAbBrutto

Currency BetragAbBrutto *(Betrag, Rundung)*

Diese Funktion errechnet den Steuerbetrag ab dem im Parameter angegebenen Brutto-Betrag, gemäss dem Inhalt des Steuer-Objekts.

Rückgabe Der Steuerbetrag.

Parameter *Betrag*: Currency

Auf diesen Betrag werden die Steuern erhoben.

Rundung: Long

Ein Rundungswert, der wie folgt definiert ist:

'floor value' * 10000. z.B für Runden auf Fr 0.01 muss der Wert 100 eingegeben werden.

Siehe Steuer, BetragAbNetto, BetragRechnen

2.25.3 Steuer.BetragAbNetto

Currency BetragAbNetto *(Betrag, Rundung)*

Diese Funktion errechnet den Steuerbetrag ab dem im Parameter angegebenen Netto-Betrag, gemäss dem Inhalt des Steuer-Objekts.

Rückgabe Der Steuerbetrag.

Parameter *Betrag*: Currency

Auf diesen Betrag werden die Steuern erhoben.

Rundung: Long

Ein Rundungswert, der wie folgt definiert ist:

'floor value' * 10000. z.B für Runden auf Fr 0.01 muss der Wert 100 eingegeben werden.

Siehe Steuer, BetragAbBrutto, BetragRechnen

2.25.4 Steuer.BetragRechnen

Currency BetragRechnen *(Betrag, Rundung)*

Diese Funktion errechnet den Steuerbetrag ab dem im Parameter angegebenen Betrag, gemäss dem Inhalt dieses Steuer-Objekts.

Rückgabe Der Steuerbetrag.

Parameter *Betrag*: Currency

Auf diesen Betrag werden die Steuern erhoben.

Rundung: Long

Ein Rundungswert, der wie folgt definiert ist:

'floor value' * 10000. z.B für Runden auf Fr 0.01 muss der Wert 100 eingegeben werden.

Siehe Steuer, BetragAbNetto, BetragAbBrutto

2.25.5 Steuer.Einfuegen

Integer Einfuegen()

Schreibt einen neuen Steuersatz in die FibuNT Steuerdatei. Der SteuerID dieses Objekts darf noch nicht von einem anderen Steuersatz verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Steuer, Schreiben

2.25.6 Steuer.Lesen

Integer Lesen *(SteuerID)*

Liest einen Steuersatz von der Datei.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *SteuerID*: String[5]

Der Kürzel des Steuersatzes.

Siehe Steuer, SteuerID

2.25.7 Steuer.LesenRel

Integer LesenRel *(Wie)*

Liest einen Steuersatz von der Datei relativ zum Steuersatz, der sich jetzt im Steuer-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie der nächste Steuersatz gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Beispiel Das Beispiel liest alle Steuersätze und zeigt sie in einer Dialog Box. Es wird angenommen, dass das Objekt Man als FibuNT geöffnet wurde.

```
Dim Steuer As Object
Set Steuer = Man.NeuSteuer()
res% = Steuer.LesenRel(0)
Do While res% = 0
```

```
MsgBox "ID: " + Steuer.SteuerID + Chr$(10) + "Konto: " +
Steuer.Konto
res% = Steuer.LesenRel(3)
Loop
```

Siehe Steuer, Lesen

2.25.8 Steuer.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Steuersatz in die Steuerdatei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Steuer, Einfuegen

2.26 Struktur

2.26.1 Struktur Objekt

Bezeichnung Mit diesem Objekt können FibuNT Kontenstruktur-Stufen/Einträge bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuStruktur erstellt.

Eigenschaften AbstandNachher: Integer

Diese Eigenschaft legt fest, wieviel Platz auf einer Auswertung noch vorhandenden sein muss, damit dieser Struktureintrag noch auf die gleiche Seite kommt. Einheiten sind in 'Twips'.

AbstandVorher: Integer

Der Abstand der vor diesem Struktureintrag auf einer Auswertung gemacht wird.

Ansichten: Long

Flags die festlegen, zu welchen Ansichten dieser Struktureintrag gehört.

AnzDaten: Long (nur Lesen)

Die Anzahl der Datensätze in der aktuellen Kontenstruktur-Datei.

AnzHaben: Long (nur Lesen)

Die Anzahl der Haben-Buchungen dieser Stufe.

AnzSoll: Long (nur Lesen)

Die Anzahl der Soll-Buchungen dieser Stufe.

BetragHaben: Currency (nur Lesen)

Die Summe der Haben-Buchungen dieser Stufe.

BetragHabenFW: Currency (nur Lesen)

Die Summe der Fremdwährungs Haben-Buchungen dieser Stufe.

BetragSoll: Currency (nur Lesen)

Die Summe der Soll-Buchungen dieser Stufe.

BetragSollFW: Currency (nur Lesen)

Die Summe der Fremdwährungs Soll-Buchungen dieser Stufe.

Bezeichnung: String[31]

Die Bezeichnung dieser Stufe. Ist diese leer, dann wird in FibuNT die aktuelle Bezeichnung des Kontos verwendet, auf das in diesem Eintrag verwiesen wird. (Siehe Struktur.KontoNr)

Bezeichnung2: String[31]

Die zweite Zeile der Bezeichnung dieser Stufe.

Budget : Currency (nur Lesen)

Das Budget dieser Stufe.

Budget2: Currency (nur Lesen)

Das Budget dieser Stufe für das übernächste Jahr.

CompareID: String[13]

Der ID der Struktureintrags mit dem Verglichen wird, wenn es um eine Auswertung geht, bei der Prozentangaben gemacht werden.

FlagNeueSeite: Boolean

Diese Eigenschaft ist 'True', wenn vor diesem Struktureintrag, bei Auswertungen ein Seitenumbruch eingefügt werden soll.

HatSpiegelKonto: Boolean (nur Lesen)

Diese Eigenschaft ist 'True', wenn das Konto mit einem Spiegelkonto verbunden ist.

IstHaben: Boolean

Wenn diese Eigenschaft 'True' ist, wird das Vorzeichen des Saldos in FibuNT in der Bilanz oder Erfolgsrechnung gewechselt. Es hat nichts mit der Berechnung der Saldi zu tun.

IstLeer: Boolean

Diese Eigenschaft ist 'True', wenn das Konto, auf das in diesem Eintrag verwiesen wird leer ist (Siehe Konto.IstLeer).

KeyId: String[13] (Index)

Diese in FibuNT als 'Key' bezeichnete Eigenschaft hat, je nach Struktur.Typ des Struktureintrags mehrere Bedeutungen. Für die Konto oder Spiegel-Konto Typen ist es die Konto-Nr eines Kontos. In diesem Fall darf diese Eigenschaft kein leerer String sein. Für die anderen Typen ist es ein beliebiger Code, der in den Auswertungen ausgegeben werden kann, oder der als eindeutige Identifizierung diese Eintrags benützt werden kann. Achtung! Diese Eigenschaft muss für die einzelnen Typen, pro Eintrag, eindeutig sein, sonst meldet die Funktion Struktur.Schreiben den Fehler 5.

KeyId2: String[13] (Index)

Eine Alias variante des KeyId.

KeyNr: Long (nur Lesen)

Die KeyNr des aktuellen Struktureintrags.

KeyBruder: Long (nur Lesen)

Die KeyNr des Struktureintrags des Bruders.

KeySohn: Long (nur Lesen)

Die KeyNr des Struktureintrags des Sohnes.

KeyVater: Long (nur Lesen)

Die KeyNr des Struktureintrags des Vaters.

PlanCount: Integer (nur Lesen)

Diese Eigenschaft ist für den internen Gebrauch reserviert.

Saldo: Currency (nur Lesen)

Der Saldo dieser Stufe. Dieser Betrag ist gleich wie Struktur.Vortrag + Struktur.BetragSoll - Struktur.BetragHaben

SaldoFW: Currency (nur Lesen)

Der Saldo dieser Stufe in der Fremdwährung. Dieser Betrag ist gleich wie Struktur.VortragFW + Struktur.BetragSollFW - Struktur.BetragHabenFW

Stufe: Integer (nur Lesen)

Die Stufe des aktuellen Struktureintrags.

TotalZähler: Integer

Diese Eigenschaft wurde durch Struktur.Zaehler1 und Struktur.Zaehler2 ersetzt.

Typ: Integer

Der Typ des Struktureintrags. Folgendes sind zulässige Typen:

0 - Hilfs-Gruppe

1 - Bilanz-Gruppe (nur verwendbar für die höchste Stufe)

2 - Erfolgs-Gruppe (nur verwendbar für die höchste Stufe)

3 - Konto

4 - Spiegel-Konto

5 - Zähler

Unsichtbar: Boolean

Der Unsichtbar-Flag, der die Anzeige dieses Eintrags in Auswertungen in der Fibu unterdrückt.

Vergleichen: Boolean

Bei Auswertungen mit Vergleichen soll bei diesem Record kein Vergleich gemacht werden, wenn diese Eigenschaft auf FALSE gesetzt wird.

Vorjahr: Currency (nur Lesen)

Der Vorjahressaldo dieser Stufe.

Vortrag: Currency (nur Lesen)

Der Saldovortrag dieser Stufe.

VortragFW: Currency (nur Lesen)

Der Saldovortrag dieser Stufe in der Fremdwährung.

WurzelTyp: Integer

Diese Eigenschaft gilt nur für Struktureinträge, die einen Struktur.Typ Gruppe haben und in der höchsten Stufe sind. Gruppen mit diesem Typ sind eine Art spezieller Einstiegs-Punkt, oder Fixpunkt für bestimmte Auswertungen. Folgende Wurzel-Typen sind definiert:

0 - Keine besondere Gruppe

1 - Fixpunkt für die Bilanz

2 - Fixpunkt für die Erfolgsrechnung

3 - Fixpunkt für die Investitionsrechnung

4 - Fixpunkt für Kostenstellenauswertungen.

Zaehler1: String[13]

Der Id des Struktureintrags (TotalZählers), der mit diesem Struktureintrag verbunden ist. Ein leerer String bedeutet: keinen Zähler.

Zaehler2: String[13]

Der Id des Struktureintrags (TotalZählers), der mit diesem Struktureintrag verbunden ist. Ein leerer String bedeutet: keinen Zähler.

ZaehlerTyp1: Integer

Die Art wie die Beträge vom diesem Struktureintrag zum Zähler übertragen werden. Folgende Typen sind möglich:

1 - Werte werden zum Zähler addiert.

0 - Werte werden vom Zähler abgezogen.

ZaehlerTyp2: Integer

Die Art wie die Beträge vom diesem Struktureintrag zum Zähler übertragen werden. Folgende Typen sind möglich:

1 - Werte werden zum Zähler addiert.

0 - Werte werden vom Zähler abgezogen.

2.26.2 Struktur.Einfuegen

Integer Einfuegen *(KeyNr, Wie)*

Fügt den Inhalt dieses Struktur-Objekts in die Strukturdatei ein. Nur die Eigenschaften, die auch in FibuNT gesetzt werden können, werden verwendet, der Rest der Eigenschaften werden automatisch initialisiert. z.B Beträge, Stufe, Verknüpfungen, usw.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Hier sind einige der häufigeren Fehlerstatus:

4 - Der Eintrag mit der angegebenen KeyNr ist nicht vorhanden.

5 - Es besteht schon ein Eintrag mit dem selben KeyId.

88 - Es wird gerade mit FibuNT in der Struktur gearbeitet.

Parameter *KeyNr*: Long

Die KeyNr der Stufe, die gelesen werden soll.

Wie: Integer

Die Art wie die Struktur eingefügt werden soll:

0 : Den Struktureintrag auf gleiche Stufe (Bruder) einfügen.

1 : Den Struktureintrag auf einer neuen Stufe (Son) einfügen.

Kommentar Diese Funktion überprüft nicht, ob das Konto auch wircklich Vorhanden ist, wenn ein Struktureintrag für ein Konto eingefügt wird. Das eigentliche Konto sollt vorher in die Konto-Datei eingefügt werden, damit Benutzer die gerade an der Fibu arbeiten keine Fehlermeldungen bekommen.

Beispiel Das Beispiel zeigt wie eine neue Kostenstelle in die Struktur eingefügt wird. Beleg gelesen, verändert und dieser anschliessend geschrieben wird.

Keine Fehlerüberprüfung:

```
' Mandant erstellen und öffnen
Dim Man As Object
Set Man = CreateObject("FibuNT.Mandant")
Man.Login 1, "C:\ProgramData\Sage\Data\Rewe\SageDemo16"

' Struktur Objekt erstellen und lesen
Dim Struktur As Object
Set Struktur = Man.NeuStruktur()
res% = Struktur.Lesen2("900", 0)
' wir brauchen nur die KeyNr zu merken
KeyNr& = Struktur.KeyNr

' Einige Eigenschaften setzen (Typ Konto/Kst)
Struktur.Typ = 3
Struktur.KeyId = "9005"
Struktur.Bezeichnung = ""
Struktur.IstHaben = False

' Und als erstes Konto der Gruppe einfügen ...
res% = Struktur.Einfuegen(KeyNr&, 1)
```

Siehe Struktur, Lesen2, KeyId

2.26.3 Struktur.GetCodeEx

Long GetCodeEx *(Index)*

Diese Funktion ist für den internen Gebrauch reserviert.

Rückgabe Ein Long-Wert.

Parameter *Index*: Integer

Der Index des gewünschten CodeEx-Teils.

2.26.4 Struktur.Leeren

Void Leeren()

Diese Funktion setzt alle Eigenschaften des Struktur-Objekts auf seine Standard-Werte.

Rückgabe keine

Siehe Struktur

2.26.5 Struktur.Lesen

Integer Lesen *(KeyNr)*

Liest einen Struktureintrag von der Datei, wenn Sie die KeyNr kennen.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *KeyNr*: Long

Die KeyNr der Stufe, die gelesen werden soll.

Siehe Struktur, Lesen2, LesenRel

2.26.6 Struktur.Lesen2

Integer Lesen2 *(KeyId, Typ)*

Liest einen Struktureintrag von der Datei, wenn Sie den KeyId, oder eine Konto-Nr kennen.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *KeyId*: String

Der KeyId des zu lesenden Datensatzes.

Typ: Integer

Der gewünschte Typ des Struktureintrages. Folgende Typen sind definiert:

0 : Gruppe

3 : Konto

4 : Spiegel-Konto

5 : Zähler

Siehe Struktur, KeyId, Lesen, LesenRel

2.26.7 Struktur.LesenRel

Integer LesenRel (Wie)

Liest einen Struktureintrag von der Datei, relative zum Eintrag, der sich jetzt schon in diesem Struktur-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst ein Fehlerstatus.

Parameter *Wie*: Integer

Die Art wie die nächste Stufe die gelesen werden soll:

0 : Liest den ersten Struktureintrag aus der Datei.

1 : Liest diesen Struktureintrag wieder, gemäss KeyNr.

2 : Liest den Vater dieses Struktureintrags.

3 : Liest den Bruder dieses Struktureintrags.

4 : Liest den Sohn dieses Struktureintrags.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei.

Siehe Struktur, Lesen, Lesen2

2.26.8 Struktur.Schreiben

Integer Schreiben()

Schreibt den zuvor gelesenen Struktureintrag in die Strukturdatei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Struktur

2.26.9 Struktur.SetCodeEx

Void SetCodeEx *(Index, Value)*

Diese Funktion ist für den internen Gebrauch reserviert.

Rückgabe keine

Parameter *Index*: Integer

Der Index eines CodeEx-Teils.

Value: Long

Der Wert des CodeEx-Teils.

2.26.10 Struktur.Wechseln

Long Wechseln *(PlanNr)*

Mit dieser Funktion können Sie dieses Objekt mit einem den Neben- Kontenplänen verbinden.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Parameter *PlanNr*: Integer

Die Nummer des Kontenplans, mit dem dieses Objekt verbunden werden soll.

Kommentar Die Funktion überprüft, ob der Kontenplan für dieser Mandant aktiv ist (nicht gesperrt).

2.27 Waehrung

2.27.1 Waehrung Objekt

Bezeichnung Mit diesem Objekt können FibuNT Fremdwährungen bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuWaehrung erstellt.

Eigenschaften Konto: String[13]

Das Fibu-Konto auf das ein Valutaausgleich gebucht werden soll. Wenn Sie diese Eigenschaft setzen, müssen Sie darauf achten, dass das Fibu-Konto gültig ist.

Kurs: Single

Der Kurs dieser Währung.

Einheit: Integer

Ein Quotient durch den der Kurs geteilt wird, bevor er angewendet wird.

ID: String[5]

Der eindeutige Kürzel der Fremdwährung.

2.27.2 Waehrung.Einfuegen

Integer Einfuegen()

Schreibt eine neue Fremdwährung in die FibuNT Währungsdatei. Der ID dieses Objekts darf noch nicht von einer anderen Fremdwährung verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Waehrung, Schreiben

2.27.3 Waehrung.Lesen

Integer Lesen(ID)

Liest eine Fremdwährung von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *ID*: String[5]

Der Kürzel der Frendwährung.

Siehe Waehrung, ID

2.27.4 Waehrung.LesenRel

Integer LesenRel *(Wie)*

Liest ein Fremdwährung von der Datei relativ zur Währung, die sich jetzt im Waehrung-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Fremdwährung gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Beispiel Das Beispiel liest alle Währungen und zeigt sie in einer Dialog Box. Es wird angenommen, dass das Objekt Man als FibuNT geöffnet wurde.

```
Dim Waehrung As Object
Set Waehrung = Man.NeuWaehrung()
res% = Waehrung.LesenRel(0)
Do While res% = 0
MsgBox "ID: " + Waehrung.ID + Chr$(10) + "Konto:" +
Waehrung.Konto
res% = Waehrung.LesenRel(3)
Loop
```

Siehe Waehrung, Lesen

2.27.5 Waehrung.Loeschen

Integer Loeschen()

Entfernt die zuvor gelesene Waehrung aus der Datei.

Rückgabe 0 wenn erfolgreich gelöscht, sonst einen Fehlerstatus.

Siehe Waehrung, LesenRel

2.27.6 Waehrung.Schreiben

Integer Schreiben()

Schreibt die zuvor gelesene Fremdwährung in die Währungsdatei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Waehrung, Einfuegen

2.28 Zahlbed

2.28.1 Zahlbed Objekt

Bezeichnung Mit diesem Objekt können DebiNT und KrediNT Zahlungsbedingungen bearbeitet werden. Dieses Objekt wird mit der Funktion Mandant.NeuZahlbed erstellt.

Eigenschaften ID: String[5]

Der eindeutige Kürzel der Zahlungsbedingung.

LimitErloes: Currency

Der Grenz-Betrag beim dem ein OP abgeschlossen wird.

LimitMahn: Currency

Der Grenz-Betrag ab dem gemahnt werden soll, wenn nötig.

MahnGebuer1: Currency

Die Mahngebüren der 1. Mahnung.

MahnGebuer2: Currency

Die Mahngebüren der 2. Mahnung.

MahnGebuer3: Currency

Die Mahngebüren der 3. Mahnung.

MahnGebuer4: Currency

Die Mahngebüren der 4. Mahnung.

MahnTage1: Integer

Die Anzahl der Tage bis zur 1. Mahnung.

MahnTage2: Integer

Die Anzahl der Tage bis zur 2. Mahnung.

MahnTage3: Integer

Die Anzahl der Tage bis zur 3. Mahnung.

MahnTage4: Integer

Die Anzahl der Tage bis zur 4. Mahnung.

Skonto1: Single

Der gewährte Skonto-Prozentsatz, wenn die erste Frist eingehalten wird.

Skonto2: Single

Der gewährte Skonto-Prozentsatz, wenn die zweite Frist eingehalten wird.

Tage1: Integer

Die Anzahl der Tage in der Skonto1 gewährt wird.

Tage2: Integer

Die Anzahl der Tage in der Skonto2 gewährt wird.

TageNetto: Integer

Die Anzahl der Tage bis der OP fällig wird.

Text: String[31]

Die Text oder Bezeichnung der Zahlungsbedingung.

2.28.2 Zahlbed.Einfuegen

Integer Einfuegen()

Schreibt eine neue Zahlungsbedingung in die Datei. Der ID dieses Objekts darf noch nicht von einer anderen Zahlungsbedingung verwendet worden sein.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Zahlbed, Schreiben

2.28.3 Zahlbed.Lesen

Integer Lesen *(ID)*

Liest eine Zahlungsbedingung von der Datei in dieses Objekt.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *ID*: String[5]

Der Kürzel der Zahlungsbedingung.

Siehe Zahlbed, ID

2.28.4 Zahlbed.LesenRel

Integer LesenRel *(Wie)*

Liest eine Zahlungsbedingung von der Datei relativ zur Zahlungsbedingung, die sich jetzt im Zahlbed-Objekt befindet.

Rückgabe 0 wenn erfolgreich, sonst einen Fehlerstatus.

Parameter *Wie*: Integer

Die Methode wie die nächste Zahlungsbedingung gelesen werden soll:

0 (FIRST): Liest den ersten Datensatz der Datei.

1 (EQUAL): Liest den gleichen Datensatz wieder, gemäss Primärschlüssel.

2 (LESS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadLess.

3 (GREATER): Liest den nächsten Datensatz mit Hilfe der Operation ReadGreater.

10 (PREVIOUS): Liest den vorhergehenden Datensatz mit Hilfe der Operation ReadPrevious.

11 (NEXT): Liest den nächsten Datensatz mit Hilfe der Operation ReadNext.

Kommentar Das Lesen mit Wie = 0 ist nicht relativ zu diesem Objekt, sondern zur ganzen Datei. Das Lesen mit Wie = 10 und Wie = 11 ist nicht relativ zu diesem Objekt, sondern zur letzten Lese-Operation.

Beispiel Das Beispiel liest alle Zahlungsbedingungen und zeigt sie in einer Dialog Box. Es wird angenommen, dass das Objekt Man als DebiNT- oder KrediNT-Mandantgeöffnet wurde.

```
Dim Zab As Object
Set Zab = Man.NeuZahlbed()
res% = Zab.LesenRel(0)
Do While res% = 0
MsgBox "ID: " + Zab.ID + Chr$(10) + "Text: " + Zab.Text
res% = Zab.LesenRel(3)
Loop
```

Siehe Zahlbed, Lesen

2.28.5 Zahlbed.Schreiben

Integer Schreiben()

Schreibt die zuvor gelesene Zahlungsbedingung in die Datei.

Rückgabe 0 wenn erfolgreich geschrieben, sonst einen Fehlerstatus.

Siehe Zahlbed, Einfuegen

2.28.6 Zahlbed.Skonto

Single Skonto (OP-Datum, Zahl-Datum)

Diese Methode ermittelt ob Skonto1, Skonto2 oder gar kein Skonto erlaubt werden soll.

Rückgabe Der Skonto-Prozentsatz Skonto1, Skonto2 oder 0.0.

Parameter *OP-Datum*: Date

Das Rechnungsdatum eines OPs.

Zahl-Datum: Date

Das Datum an dem bezahlt wird.

Siehe Zahlbed, Skonto1, Skonto2

3.0 Fehler-Codes

Hier ist die Liste der möglichen Fehlerstatus, die von Objekt-Methoden zurück gegeben werden.

>0 Wenn der Fehlerstatus positiv ist, deutet es auf einen Datenbankfehler hin.

-100 Ungültiger Parameter.

Eines der Parameter enthält einen ungültigen Wert.

-101 Mandant nicht offen.

Der Mandant wurde noch nicht geöffnet und muss offen sein, damit die betreffende Funktion erfolgreich ist.

-102 String zu lange.

Einer der String-Parameter enthält zu viele Zeichen. Sehen Sie in der Funktions-Referenz nach, wieviele Zeichen für die betreffende Funktion erlaubt sind.

-103 Ungenügend Speicher.

Die Funktion konnte nicht vollständig durchgeführt werden, weil zu wenig Arbeitsspeicher vorhanden ist.

-104 Ungültiges Objekt.

Es wurde der Funktion ein ungültiges OLE-Objekt als Parameter übergeben.

-105 Nicht implementiert.

Die Funktion wurde noch nicht implementiert.

-106 Ungültiges Datum.

Das im Parameter übergebene Datum ist ungültig.

-107 Operation abgebrochen.

Die Operation wurde vom Benutzer abgebrochen.

-108 E/A Fehler.

Ein nicht genauer spezifizierter E/A Fehler ist aufgetreten. Dieser Fehler kann z.B. auch auftreten, wenn keine weiteren Datensätze in der Datei vorhanden sind.

-109 OP Lese-Fehler.

Ein Fehler ist aufgetreten beim Lesen des benötigten OPs. Kontrollieren Sie mit DebiNT oder KrediNT, ob der OP wirklich vorhanden ist.

-110 PK Lese-Fehler.

Ein Fehler ist aufgetreten beim Lesen des benötigten PKs. Kontrollieren Sie mit DebiNT oder KrediNT, ob das PK wirklich vorhanden ist.

-111 NULL-String.

Ein leerer String wurde als Parameter übergeben für einen Parameter, der nicht leer sein darf, z.B. Eine Konto-Nr oder OP-Nr.

-112 Nicht Verfügbar.

Die Eigenschaft oder Methode dieses Objekts ist in diesem Zusammenhang, in dem sie aufgerufen wurde, nicht verfügbar.

-113 Konto Lese-Fehler.

Ein Fehler ist aufgetreten beim Lesen des benötigten Kontos. Kontrollieren Sie mit FibuNT, ob das Konto wirklich vorhanden ist.

-114 Nur AddressNT.

Die Funktion arbeitet nur mit der internen Adressverwaltung.

-115 Leerer Beleg.

Der Beleg der gebucht werden soll enthält keine Buchungen.

-116 Adress Lese-Fehler.

Ein Fehler ist aufgetreten beim Lesen der benötigten Adresse. Kontrollieren Sie mit DebiNT/KrediNT, ob die Adresse wirklich vorhanden ist.

-117 K-Plan gesperrt

Der Kontenplan ist gesperrt, oder die Kontenplan-Nr ist ungültig.

-118 Keine Buchungsmasken

Es wurden keine Buchungsmasken in die Registry eingetragen.

-119 Nur für Vorerfassung

Diese Funktion arbeitet nur mit vorerfassten Belegen.

-120 Unlöschbar

Der Datensatz kann nicht gelöscht werden, weil Buchungen oder andere Datensätze darauf verweisen.

-121 Kein Hyparchiv gefunden

Die Archivierungsschnittstelle ist als installiert in der Registry eingetragen, konnte aber nicht erstellt werden. Versuchen Sie in den Sage 50 Applikationen (sprich Kredi) einen OP zu archivieren.

-122 OP schon archiviert

Der OP im Beleg ist schon archiviert. Falls Sie den OP im Archiv überschreiben möchten, müssen Sie die Methode nochmals aufrufen nachdem Sie mit SetArchivModus den Modus auf Überschreiben gesetzt haben.

-123 OP kann in Archiv nicht überschrieben werden

Der OP konnte im Archiv nicht überschrieben werden.

-124 Unbekannter Fehler beim Archivieren

Es trat beim Archivieren mit Hyparchiv ein unbekannter Fehler auf.

-125 Beleg ist nicht ausgeglichen

Ein Beleg kann nur gespeichert werde, wenn er ausgeglichen ist. Dieser Fehler tritt häufig ein, wenn die Leitwährung über die Fremdwährung errechnet wird und am Schluss nicht gerundet wird. Beträge werden generell 4-Stellig hinter dem Komma gespeichert. Es kann also zu Soll/Haben Differenzen an der 3. und 4. Kommastelle kommen, wenn nicht auf 1 oder 5 Rappen gerundet wird.

-128 Benutzer kat keinen Zugriff auf Mandanten

Wenn die Benutzerverwaltung auf Mandantenebene aktiv ist, kann ein Mandant nur geöffnet werden, wenn der Benutzer auch Zugriff hat.

-129 DMS Fehler

Ein unerwarteter Fehler beim Aufrufen des Document Management Systems (DMS) ist aufgetreten. Möglicherweise ist kein DMS auf diesem System installiert.

-130 Obsolete

Die Funktion wird nicht mehr verwendet.

-131 ESR/Referenz ungültig

Der ESR Text oder die Referenz-Nr ist ungültig.

-132 BVId ungültig

Der BVId darf nicht leer sein, um die Bankvebindung zu schreiben.

-133 IBAN-Definitionsdatei nicht gefunden.

Die Defintionsdatei "IBANDef.ini" muss am gleichen Ort die das SOK liegen.

-134 Unbekannter Ländercode

Die IBAN enthält einen unbekannten Ländercode

-135 Ungültige Zeichen

Die IBAN enthält Ungültige Zeichen. Leerschläge sind nie erlaubt, Buchstaben sind je nach Land erlaubt.

-136 Ungültige IBAN-Nummer

Die errechnete Prüfziffer stimmt nicht mit der der IBAN-Nummer überein.

-137 Die Länge der IBAN-Nummer stimmt nicht

Je nach Landesspezifikationen müssen die Konto- und Clearing- Felder bestimmt Längen haben.

-138 Ein DAL Mandant Objekt konnte nicht erstellt werden.

Es konnte kein DAL Mandant erstellt werden.

-139 Ein Login auf einen Mandanten konnte nicht ausgeführt werden.

Das Login eines DAL Mandanten ist fehlgeschlagen