ADDISON

The Software to manage your Profit

addison | ADDISON Connect

Schnittstellenentwicklung

- ☐ Produktinfo
- ☑ **Produktdokumentation**
- ☐ Installation

ADDISON Connect Schnittstellenentwicklung
Benutzerhandbuch

Stand: Oktober 2006

ADDISON Software und Service GmbH
Stuttgarter Str. 35
71638 Ludwigsburg
http://www.addison.de

Inhaltsverzeichnis

	Weiterführende Informationen
	Wichtige Hinweise
	Handlungsanweisungen
	Nützliche Tipps und Tricks
	Beispiele zu den Themen

1. Allgemeiner Überblick

1.1. Allgemeines

Programmzweck

ADDISON Connect ist eine Sammlung von Schnittstellen zum Zugriff auf Programmfunktionen der ADDISON Software über Programmierwerkzeuge.

Hauptbestandteil sind Zugriffsschnittstellen auf Basis verbreiteter Middleware, unter Microsoft Windows bevorzugt auf Basis von COM (Compound Object Model von Microsoft). Darüber hinaus gehören zu den „ADDISON Connect"-Funktionen Definitionen von Erweiterungsbibliotheken, über die den ADDISON Programmen standardisierte Zusatzfunktionalität (in der Regel Schnittstellen zur Fremdprogrammen) hinzugefügt werden kann.

Den Kern der ADDISON Anwendungen bilden so genannte Business Objects. Das sind die fachlichen Funktionen der Software getrennt von der „Grafischen Benutzeroberfläche" (dem GUI).

Diese fachlichen Funktionen implementieren alle Zugriffe auf die Datenbank, die fachlichen Regeln und Abläufe innerhalb der Software. Aufgrund der Vielzahl der Zusammenhänge und der Komplexität des Datenmodells ist ein direkter Zugriff auf die Datenbank mit Programmierwerkzeugen nicht sinnvoll. Stattdessen bietet die „ADDISON Connect"-Schnittstelle eine abstraktere Zugriffsfunktion unter Berücksichtigung der vollständigen Anwendungslogik.

1.2. Überblick über die Bestandteile

1.2.1. Allgemeine Funktionen

Was benötigen Sie zur Programmentwicklung?

„ADDISON Connect"-Anwendungen können in einer Vielzahl von Programmiersprachen und Anwendungsumgebungen durchgeführt werden, z.B.:

- Microsoft Visual Basic
- Microsoft Visual Basic für Applikationen
- Microsoft C++
- Microsoft Access
- Microsoft Foxpro
- Microsoft Visual Studio.Net
- Borland C++ Builder
- Borland Delphi

Entwicklungsumgebung

Die Entwicklungsumgebung muss nur unter MS Windows ablauffähig und in der Lage sein, COM-Objekte anzusprechen.

Da hier nicht im Einzelnen die Art und Weise des Zugriffs auf die Objekte in den einzelnen Programmiersprachen verwiesen werden kann, sei hier auf die Dokumentationen der Entwicklungsumgebung verwiesen.

Beispiele

Beispiele in dieser Dokumentation beziehen sich i.d.R. auf „Visual Basic" oder „Visual Basic für Applikationen".
Da wir davon ausgehen, dass die meisten Firmen im Besitz einer Office-Lizenz sind, haben wir die wichtigsten Beispiele in Excel und Visual Basic für Applikationen realisiert, damit Sie diese selbst ausprobieren können.

Wie beginnen?

Voraussetzung ist ein installiertes ZMiS mit ADDISON Connect (aconnect.exe). Die Schnittstelle wird im Standardfall automatisch installiert und ist auf Rechnern mit installiertem ZMiS sofort verfügbar.

Die folgenden Beschreibungen basieren alle auf der Visual Basic Syntax, können aber in ähnlicher Form in andere Entwicklungsumgebungen übertragen werden. Bitte schauen Sie zu diesem Zweck im Abschnitt „Entwicklungsumgebungen" nach.

2. Bestandteile von ADDISON-Connect

2.1. Basisfunktionen

AddBOManager

Der BOManager (Business Object Manager) ist die zentrale Instanz zum Laden der Runtime-Umgebung der Business Objects innerhalb von ZMiS. Zum Zugriff auf die BO-Funktionen muss ein Objekt des Typs AddBOManager angelegt werden und ein die Funktion Connect ausgeführt werden.

Durch den Aufruf von „Connect" wird die BO-Umgebung (DLLs und Initialisierungsdateien) geladen und der Zugriff zur Datenbank hergestellt. Optional wird beim Verbinden ein Logindialog eingeblendet (kann ausgeblendet werden).
Bis zum Aufruf der Methode Close bleibt der BO-Manager geöffnet, auch wenn man das Objekt BO-Manager wieder verwirft.

Tipp

Es ist nicht günstig, bei jedem Zugriff die Runtime-Umgebung des ZMiS neu zu laden. Daher implementieren wir den BO-Manager so, dass er bis zum Close die Verbindung offen hält.
Es ist daher erforderlich, vor dem ersten Zugriff den BO-Manager zu laden und zu initialisieren (Idealerweise zum Programmstart).
Sie verbinden zu diesem Zweck das Projekt mit den COM-DLLs (in VB nennt sich das „Verweis" im Menüpunkt „Extras\Verweise"). Nach dem Registrieren der COM-DLLs erscheinen sie dort unter „ADDISON BO Library 1.0 (Basics)" und „ADDISON BO Library 1.0 (Rechnungswesen)". Damit werden die VB-Projekte mit diesen DLLs „verknüpft". Unter C++ gibt es äquivalente Verfahren (siehe Beispiele). Achten Sie auf eine korrekte Fehlerbehandlung, Sie ersparen Ihrem Systemverwalter auf Dauer viel Kopfzerbrechen.
Anschließend fügen Sie an geeigneter Stelle sinngemäß folgenden Code ein:

Beispiel Initialisierung

```
Dim pBOMAN As AddBOManager
Set pBOMAN = New AddBOManager
pBOMAN.Connect
...
' der bomanager hält intern einen Lock, am Ende der
funktion kann man ' das Interface mit folgenden Zeilen
wegwerfen
set pBOMAN = Nothing
```

Beispielsweise können Sie das in der Initialisierung des Hauptfensters Ihrer Applikation tun oder in der Applikationsinstanz ihrer MFC-Applikation (InitInstance ...). Sie können auch den BO-Manager beim ersten Zugriff öffnen und

bei jedem weiteren Zugriff mit IsOpen pollen, ob der BO-Manager noch offen ist, sowie beim Schließen wenn der BO-Manager IsOpen zurückliefert, Disconnect aufrufen.
Das Objekt AddBOManager selbst ist nur ein Wrapper für den eigentlichen BO-Manager. Solange Sie nicht Connect aufrufen, verfügt die Klasse auch nicht über eine gültige Instanz und wird jeden Zugriff ablehnen. Auch wenn Sie die Instanz des AddBOManagers zerstören, wird die eigentliche BO-Manager-Instanz bis zum Disconnect weiterleben. (es kann allerdings sein, dass die Instanz automatisch rausgeworfen wird, beispielsweise wirft der Microsoft Transaktion Server die Applikationsobjekte von Zeit zu Zeit automatisch weg, bei VB oder C++ selbst gibt es solche Automatismen nicht).

Vorschlag zum Zugriff

Ein Vorschlag zum Zugriff wäre der folgende Programmcode:

```
Function getBOManager() As AddBOManager
  Dim boman As New AddBOManager
  If boman.IsOpen Then
    Set getBOManager = boman
  Else
    boman.Connect
    If boman.IsOpen Then
      Set getBOManager = boman
    Else
      MsgBox "Fehler beim Öffnen des BOManagers"
      Set getBOManager = Nothing
    End If
  End If
  Set boman = Nothing
End Function
```

In diesem Fall
Vergleichen Sie die Verwendung innerhalb der Testanwendung „testaconnect.xls“.
Vom BOManager aus laden Sie auch die Mandanten. Sie können über die Methode „GetMandant“ und die Mandantennummer ein Objekt vom Typ AddMandant erstellen.

Adressen

Zentraler Einstiegspunkt ist der Mandant. Am Mandanten „hängen“ die Projekte mit ihren Daten (wie Finanzbuchhaltung, Lohn und Kostenrechnung).

Alle diese Daten sind als „Objekte“ definiert und können über Business Objects dargestellt werden. Dabei gibt es eine ganze Reihe von Objekten, an denen eine oder mehrere Adresse verwaltet wird, beispielsweise Mandant oder Kundenkonto. Auf die Adressdaten dieses Objekts kann mit Hilfe der Klasse AddGeschaeftspartner zugegriffen werden.

Die Methode bietet umfangreiche Möglichkeiten, die Adressdaten abzufragen.

Ein beispielhafter Datenzugriff ist in dem Modul „modDoIt" der Anwendung „testaconnect.xls" zu finden.

Data Warehouse

Der Zugriff auf das Data Warehouse ist über die Basisfunktionen realisiert. Vergleichen Sie dazu den Zugriff, wie er innerhalb des Moduls modDWView realisiert ist.

Die einfachste Form des Zugriffs erfolgt über die Klasse AddWuerfelAnsicht. Mittels dieser Klasse kann man in einfacher Form auf die im Data Warehouse definierten Ansichten zugreifen.

Man kann auch auf die Datenfunktionen direkt über die Klasse AddWuerfel zugreifen.

Weitere Klassen

Objekt	Beschreibung
AddMandant	Mandantenklasse: von hier aus kann man auf die ZMiS-Daten und auf die Projekte zugreifen
AddGeschaeftspartner	Implementierung des grundlegenden Adressobjektes: sowohl die ZMiS-Daten als auch die Daten der Fibu-Personenkonten.
AddWaehrung	Währungsobjekt: Euroumrechnung usw.
AddProject	Projektzugriff: Mittels dieses Objekts erfolgt der Zugriff auf die Extension-DLLs.
AddKanzlei	Kanzleiverwaltung (Steuerberaterspezifisch)
AddProtokoll	Zugriff auf die anfallenden Messages. Die Applikation befindet sich immer im Batchmodus. Alle Meldungen der Programme werden in temporäre Dateien geschrieben und können mit diesem Objekt abgefragt werden.
AddAdressVerwaltungMandant	Suche von Adressen innerhalb eines Mandanten
AddWorkflowManager	Verwaltung der Ordner für den Office Manager innerhalb von ZMiS, Anzeige der Ordner und der Einträge, Hinzufügen von neuen Einträgen
AddWorkflowItem	Ein einzelner Eintrag innerhalb des Workflowmanagers

2.2. Rechnungswesen

Klassenübersicht

Um auf die Rechnungswesenklassen zuzugreifen, muss das Modul afibu.dll innerhalb des Projekts referenziert sein.

Objekt	Beschreibung
AddFibuProjectImp	Projektklasse Fibu, von hier kann man die Wirtschaftsjahre

Objekt	Beschreibung
	laden usw.
RwFibuWJ	Ein Wirtschaftsjahr, von hier kann man beispielsweise die Wirtschaftsjahre laden
RwBilanzInfo	-
RwBilanzItem	-
RwSachkonten	Sachkontenverwaltung
RwDebitorkonten	Kundenkontenverwaltung
RwKreditorkonten	Kreditorenkontenverwaltung
RwFibuKonto	Konto: Saldoabfrage, bei Personenkonto Geschäftspartnerobjekt (Adresse), Kreditlimit ...
RwOPSuche	Heraussuchen der OPs (nach Konto, nach Belegnummer, nach Betrag)
RwOP	ein OP: Abfrage der OP-Daten
RwBuchungsImport	Aufruf der Importfunktionen (WINFIB, Datevformat ...)

Zugriff

Der Zugriff erfolgt vom Mandanten mit folgenden Codezeilen:

```
Dim myaddmandant As addmandant
Set myaddmandant = boman.GetMandant(mandant)
If Not myaddmandant Is Nothing then
  if Not myaddmandant.GetFibuProject Is Nothing Then
    Dim fibuprojekt As New AddFibuProjectImp
    fibuprojekt.AttachMandantenNr (mandant)

    Dim fibuwj As RwFibuWJ
    Set fibuwj = fibuprojekt.GetLastWJ

    …
  Endif
Endif
```

Dieses Codefragment zeigt exemplarisch den Zugriff auf das letzte (aktuellste) Wirtschaftsjahrobjekt.

Vom Wirtschaftsjahrobjekt kommt man dann auf die Konten und OP-Funktionen.

Kontenverwaltung

Die Kontenverwaltung erfolgt über die Kontenobjekte (Debitorkonten, Kreditorkonten und Sachkonten). Diese Objekte werden vom Wirtschaftsjahr geladen.
FibuProjekt → Wirtschaftsjahr → Debitorkonten → Debitorkonto

Debitorkonten-Objekt

Das Debitorkonten-Objekt verfügt über Methoden zum Suchen und Durchlaufen der Konten und gibt ein Objekt vom Typ Debitor/Sach/Kreditorkonto zurück, über das man an die Daten des Kontos herankommt.

Beispiele für den Zugriff findet man in den modDoIt-Makros der testaconnect.xls, insbesondere in der Methode Sub

CallKontendaten().

Beispiel Kontoblatt

Ein Beispiel für den Kontenzugriff ist im Modul modDoIt innerhalb von testaconnect realisiert-CallKBlattdaten()). Der Weg zum Konto ist: FibuProjekt → Wirtschaftsjahr → Debitorkonten → Debitorkonto → Kontenblatt (oder ein anderes Kontenobjekt (Sach- und Debitorkonten).

```
Set fibuwj = fibuprojekt.GetLastWJ

Dim debitoren As RwDebitorkonten
Set debitoren = fibuwj.m_Debitorkonten

Dim mykonto As New RwFibuKonto
Set mykonto = debitoren.SearchFibuKonto(konto)
If Not mykonto Is Nothing And Len(mykonto.m_KontoNr) Then
On Error Resume Next

  Dim myKBlatt As New RwKontenblatt

  myKBlatt.LoadBuchungen mandant,mykonto.m_KontoNr,fibuwj.m_Beginn,1, 15

  Dim kpos As Long
  kpos = 0

  While myKBlatt.Seek(kpos) = 0
    Range("A" & pos).Select
    ActiveCell.FormulaR1C1 = kpos + 1

    Range("B" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_Konto
    Range("C" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_BelegDatum
    Range("D" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_BelegNr
    Range("E" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_BelegNr2
    Range("F" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_Betrag
    Range("G" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_BuchungsDatum

    Range("H" & pos).Select
    ActiveCell.FormulaR1C1 = myKBlatt.m_Buchungstext

    pos = pos + 1
    kpos = kpos + 1

  Wend
```

Über die Seekfunktion positioniert man sich innerhalb der selektierten Buchungsliste.

OPs

Der Zugriff auf die OPs ist innerhalb der Funktion Sub CallOPdaten() im Modul modDoIt realisiert. Die Funktion ist ähnlich dem Kontenblattzugriff, statt des Kontoblatt-Objekts wird das Objekt RwOPSuche verwendet.

Die OPs innerhalb von ADDISON sind in zwei Bereiche eingeteilt. Die historischen OPs (ausgeziffert) und die aktuelle offenen OPs.

Man kann mittels des RwOPSuche-Objekts zwischen diesen beiden Bereichen unterscheiden.

Beispielanwendung testaconnect.xls

Die Testanwendung kann auf einem Rechner mit installiertem ZMiS sehr einfach ausprobiert werden. Öffnen Sie die Datei und öffnen Sie den Reiter „Info". Über die eingefügten Schaltflächen kann man die Makros aufrufen, die Ergebnisse der Abfragen werden in weiteren Reitern übertragen (Adressen, Fibudaten usw.).

Man kann innerhalb des VBA-Editors den Sourcecode anschauen und auch debuggen, es besteht kein Projektschutz.
modImport: Zugriff auf das Import-Interface im ADDISON Importformat
modDoIt: Datenzugriffsfunktionen mit Schwerpunkt Fibu
modDWView: Datenzugriffsfunktionen auf Data Warehouse.

Importschnittstellen

Die Importschnittstelle der Finanzbuchhaltung kann ferngesteuert aufgerufen werden. Das erfolgt über die Klasse RwBuchungsImport.

Siehe Makto modImport

Beispielanwendung testimport.xls

Im Beispiel modImport ist der Zugriff auf die Importschnittstelle definiert, es wird aus den Daten aus dem Reiter „ADDISON Import" eine einfache ADDISON Import-Datei erstellt und dann importiert. Fehlermeldungen des können über AddProtokoll angezeigt werden.

2.3. Kostenrechnung und Controlling

Zugriff über Data Warehouse

Der Zugriff auf die Daten von Kostenrechnung und Controlling kann vollständig über die Data Warehouse-Funktionalität abgewickelt werden, der Datenimport erfolgt i.d.R. ja über die Fibu-Schnittstelle.

Importschnittstellen

Für einen ferngesteuerten Import existiert die Schnittstelle KostBuchungsImport. Der Import basiert auf dem Kost-Buchungsimport von ADDISON und übernimmt damit die dort beschriebenen Einstellungen bzw. wird auch so konfiguriert.

Folgendes VB-Scipt, auszuführen als wscript kostbuch.vbs, verdeutlicht die Verwendung:

```
rem Datei: kostbuch.vbs
rem Beispiel für den Kostbuchungsimport
on error resume next

set app = CreateObject("abasic.AddBOManager")
if err.number = 0 then
   if app.Connect() = 0 then
      set import = CreateObject("akost.KostBuchungsImport")
      if err.number <> 0 then
         WScript.Echo err.number
      else
         rem Mandantennummer, Importdatei, Stapelname, Sektion aus kostimp.ini
         import.DoImport 3, "C:\tmp\test.csv", "TestStapel", "Format0"
      end if
      app.Close()
   else
      WScript.Echo "Datenbank konnte nicht geöffnet werden"
   end if
else
   WScript.Echo err.number
end if
```

2.4. Anlagenbuchhaltung

Zugriff über Data Warehouse

Der Zugriff auf die Daten der Anlagenbuchhaltung kann ebenfalls über die Data Warehouse-Funktionalität abgewickelt werden.

2.5. Personalwesen

Klassenübersicht

Um auf die Klassen des Personalwesens zuzugreifen, muss das Modul alohn.dll innerhalb des Projekts referenziert sein.

Objekt	Beschreibung
Lohn	Projektklasse Lohn, von hier aus kann u.a. auf die Arbeitnehmerdaten zugegriffen werden
Personal	Ein Arbeitnehmer, von hier aus kann auf die Personalversionen zugegriffen werden.
PersonalVersion	Die Version stellt die Personal- und Abrechnungsdaten eines Abrechnungszeitraums zu Verfügung

Zugriff

Der Zugriff erfolgt vom Mandanten mit folgenden Codezeilen:

```
Dim myaddmandant As addmandant
Set myaddmandant = boman.GetMandant(mandant)
If Not myaddmandant Is Nothing then
  if Not myaddmandant.GetLohnProject Is Nothing Then
    Dim lohnprojekt As New Lohn
    lohnprojekt.AttachMandantenNr (mandant)

    …
  Endif
Endif
```

Personaldaten

Die Ermittlung der Personaldaten erfolgt vom LohnProjekt über das Personal zur Personalversion:

LohnProjekt → Personal → PersonalVersion

Über die Personalversion sind sowohl die Personalstammdaten als auch die Abrechnungswerte erfragbar.
In dem nachfolgenden Beispiel wird die Anschrift des Mitarbeiters mit der Personalnummer 2 in dem Mandanten 1 zum Abrechnungszeitraum 01.09.2003 ermittelt:

```
Dim bo As New AddBOManager
bo.Connect

If bo.IsOpen Then
   Dim mandant As AddMandant
   Set mandant = bo.GetMandant(1)

   Dim project As AddProject
   Set project = mandant.GetLohnProject

   Dim lohnprj As New ADDISONLohn.Lohn
   lohnprj.Attach project

   Dim personal As ADDISONLohn.personal
   Set personal = lohnprj.personal.AskFor(2)

   Dim version As New ADDISONLohn.PersonalVersion
   Set version = personal.version("01.09.2003")

   Dim anschrift As String
   anschrift = _
      version.Daten("Anrede") & "\n" & _
      version.Daten("Vorname") & " " & version.Daten("Name") & "\n\n" & _
      version.Daten("Strasse") & "\n" & _
      version.Daten("PLZ") & " " & version.Daten("Ort")

   Set version = Nothing
   Set personal = Nothing
   Set project = Nothing
   Set mandant = Nothing
End If

Set bo = Nothing
```

2.6. Kanzleiverwaltung

Klassenübersicht

Um auf die Kanzleiklassen zuzugreifen, muss das Modul akanzlei.dll innerhalb des Projekts referenziert sein.

Objekt	Beschreibung
AddKanzleiImp	Projektklasse Kanzlei, von hier kann man die Kanzleiadresse, Mitarbeiter, Steuerbescheide und Steuervorauszahlungen laden.
KanzSteuerbescheid	Steuerbescheid, von hier kann man u.a. Bescheiddatum, Steuerart, Veranlagungsjahr und die einzelnen Bescheidpositionen laden.
KanzBescheidposition	Bescheidposition, von hier kann man die Bezeichnung und den Betrag laden.
KanzSteuervorauszahlungen	Steuervorauszahlung, von hier kann man die einzelnen Vorauszahlungspositionen laden.
KanzVZPosition	Vorauszahlungsposition, von hier kann man den Betrag, die Fälligkeit und das Veranlagungsjahr laden.

Zugriff

Der Zugriff auf die Adresse der Kanzlei erfolgt mit folgenden Codezeilen:

```
    Dim curKanz As New AddKanzleiImp
    Dim Partner As New AddGeschaeftspartner

    curKanz.SetCurrentKanzlei (KanzNr)
    Set Partner =
curKanz.Kanzleistandort(StandortNr)

    bf$ = Partner.m_Name1
    console.Text = bf

    Set Partner = Nothing
    Set curKanz = Nothing
```

Zugriff auf den Bescheid- bzw. Vorauszahlungsdaten

Der Zugriff auf den Bescheid- bzw. Vorauszahlungsdaten erfolgt mit folgenden Codezeilen:

```
Global curKanzleiImp As New AddKanzleiImp
Dim curSteuerbescheid As KanzSteuerbescheid
Dim curVorauszahlung As KanzSteuervorauszahlung

curKanzleiImp.SetCurrentKanzlei (KanzleiNr)
Set curSteuerbescheid = curKanzlei
 Imp.GetSteuerbescheid(MandantNr,
SteuerartNr, Veranlagungsjahr)

Dim curBescheidposition As KanzBescheidposition
curSteuerbescheid.SetFristPosition
Set curBescheidposition = curSteuerbescheid.GetNext
```

Dieses Codefragment zeigt exemplarisch den Zugriff auf einen Steuerbescheid.

Mit Hilfe der Funktionen .SetFirstPosition und GetNext können die einzelnen Bescheidabgleichs der Reihe nach geladen werden.

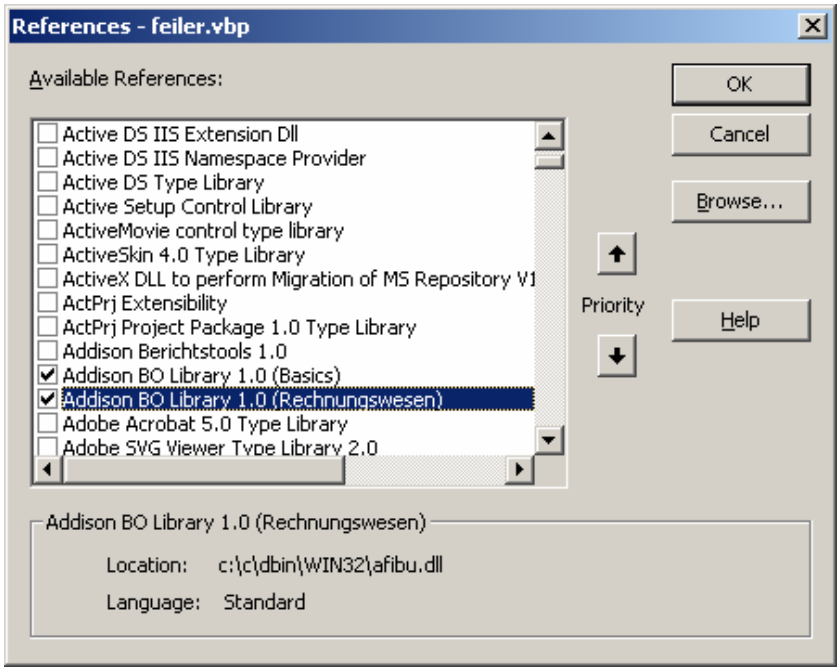
Der Zugriff auf den Steuervorauszahlungen funktioniert nach demselben Schema.

3. Entwicklungsumgebungen

3.1. Visual Basic

Integration

Die Integration in Visual Basic ist recht einfach:
Rufen Sie den Menüpunkt Project → References auf, und es erscheint der folgende Dialog. Sie aktivieren die gewünschten DLLs und die Funktionen stehen Ihnen unmittelbar als Erweiterung des VB-Sprachumfangs zur Verfügung:



In der Abbildung sind die beiden Bibliotheken für Rechnungswesen und Basisfunktionen aktiviert. Sie können die Beispiele aus der testaconnect.xls mit geringfügigen Änderungen übertragen.

Hinweise zur Anwendungsentwicklung

Eine Spielart der Einbindung ist der indirekte Zugriff über die Methode „CreateObject". Um die Abhängigkeit zwischen den ausgelieferten ADDISON-Bibliotheken und Ihrem Programmcode zu mindern, sollte der Zugriff über das Dispatch-COM-Interface realisiert werden.

In der im vorherigen Abschnitt beschriebenen Vorgehensweise wird direkt auf die ADDISON-Interfaces zugegriffen. Auch das funktioniert, es kann aber sein, dass man bei Erweiterungen der Schnittstelle die Anwendungen neu kompilieren muss.

Alle ADDISON-Objekte sind dual, unterstützen also das „generische" IDispatch-Interface und implementieren ein

spezielles Klasseninterface. Der Unterschied ist, dass bei Änderungen des Interfaces der Zugriff auf das IDispatch-Interface kompatibel bleibt, während der Zugriff auf das direkte Interface neu kompiliert werden muss.
Sollten Sie die Anbindung als Produkt einer Software ausliefern, empfiehlt sich die Implementierung über das IDispatch-Interface.

Varianten

Variante mit direktem Zugriff (IAddBOManager-Interface)

```
Dim boman as new AddBOManager
boman.Connect
```

Variante mit indirektem Zugriff (IDispatch-Interface)

```
Dim boman as Object
Set boman = CreateObject(„abasic.AddBOManager“)
boman.Connect
```

Im zweiten Fall ist es nicht notwendig (und sollte auch nicht gemacht werden) die DLL über Referenzen dem Projekt hinzufügen (Man kann in der Entwicklung mit der ersten Methode arbeiten und bei Auslieferung den Code auf die zweite Methode abändern).

Die erste Variante ist komfortabler, weil man auf die Funktionen automatisch über die Vervollständigungsfunktionen von Visual Basic arbeiten kann, die zweite Methode ist aber unabhängiger. Nutzen Sie die Software nur im Haus, kann man auch bei Updates die Anwendung einfach neu kompilieren.

3.2. Visual C++

Integration mittels Class Wizard

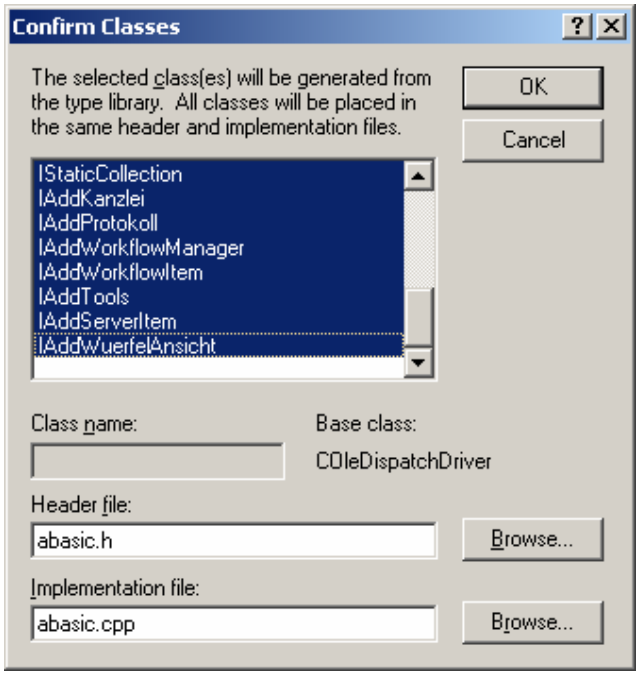
Um mit den MFC-Funktionen auf die Funkionalität von ADDISON Connect zugreifen zu können, generieren Sie in Ihrem Projekt über den Class Wizard Wrapperklassen, die den Zugriff auf das IDispatch-Interface vereinfachen.

Der Menüpunkt ist im Class-Wizard zu finden:

Dateiauswahldialog

Über „Add Class → From a type library ...“ erhalten Sie eine Dateiauswahldialog.

Suchen Sie die gewünschte DLL (z.B. die abasic.dll) heraus und öffnen diese, darauf öffnet sich der folgende Dialog:



Wählen Sie am besten alle Klassen aus und „OK". Darauf haben Sie die Wrapperklassen innerhalb von der Datei abasic.h deklariert. Sie müssen zur Verwendung diese Datei per include in ihren Code einbinden:

```
#include "abasic.h"

void TestFunktion()
{
 IAddBOManager aBOMan;
 aBOMan.CreateDispatch("abasic.AddBOManager");
 aBOMan.Connect ();
 if (aBOMan.get_IsOpen()!= 0L)
 {
        ...
        aBOMan.Close ();
 }
}
```

Die Funktionalität der Wrapperklasse liefert die MFC-Klasse COleDispatchDriver, deren Dokumentation Sie sich anschauen sollte. Das Navigieren durch den „Objektbaum" erfolgt dann über „AttachDispatch":

```
LPDISPATCH pDisp = aBoMan.GetGetMandant(1);
if (pDisp)
{
IAddMandant aMandant;
aMandant.AttachDispatch(pDisp);
...
}
```

Hinweis

Zur Verwendung der OLE-Funktionalität der MFC muss im

Member InitInstance der Applikation die Funktion AfxOleInit() aufgerufen werden, ohne diese Funktion ist die OLE-Library nicht initialisiert und funktioniert nicht.

DLLs

Die Namen der interessanten DLLs:

- abasic.dll: Basisfunktionen
- afibu.dll: Funktionen des Rechnungswesens
- akanz.dll: Funktionen der Kanzleiverwaltung
- alohn.dll: Funktionen Personalwesen.

Integration über andere Wege

Es gibt weitere Möglichkeiten, innerhalb von C++ auf die COM-Funktionen zuzugreifend.

Auf sie soll hier nur der Vollständigkeit halber eingegangen werden, sie setzen entsprechende Erfahrung in der COM-Entwicklung voraus und sind im allgemeinen schwieriger anzuwenden:

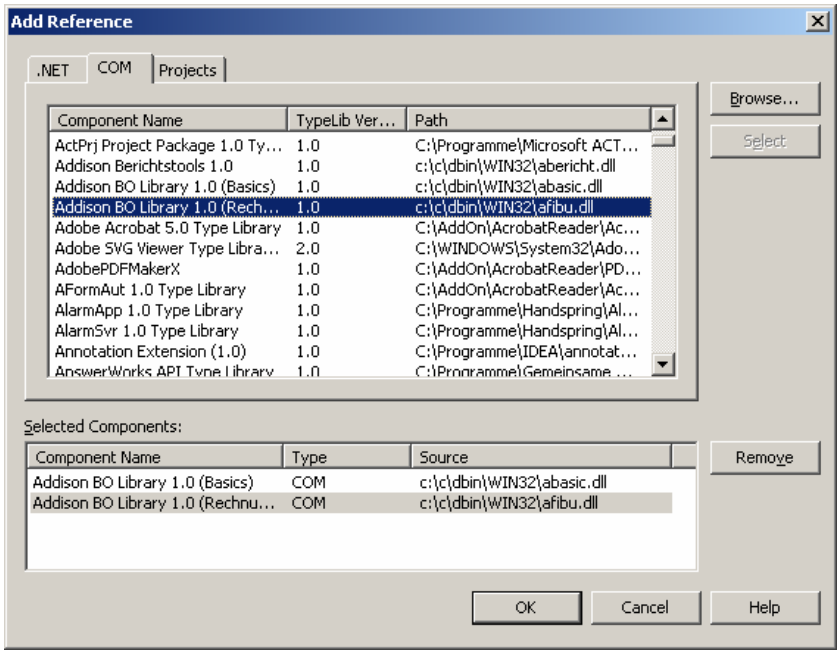
- Zugriff über das COM-API ohne Wrapperklasse
  Man kann auf die COM-Funktionen auch mit den „Low level"-API-Funktionen zugreifen (CoCreate ... usw.), oder sich ein eigenes Wrapperinterface schreiben.
- Zugriff über das #import Statement
  Dabei handelt es sich um eine Microsoft-spezifische Spracherweiterung, bei der die Typelibrary über das #import-Statement direkt eingebunden wird, ohne erst einen Wrapper zu erzeugen.
- Zugriff über ATL-Funktionen (Active Template Library)

Darüber hinaus gibt es auch weitere auf dem Softwaremarkt verfügbare freie und kommerzielle Implementierungen zum Zugriff auf COM.

3.3. Visual Studio.Net

Integration in Visual Basic.Net

Der Dialog zum Hinzufügen von Referenzen/Bibliotheken zu einem Visual Basic-Projekt hat sich erweitert. Der Befehl befindet sich jetzt unter „Project → Add Reference ...", darauf öffnet sich der folgende Dialog, wählen Sie den Reiter „COM" aus, dann können Sie die Bibliotheken wie unten zu sehen auswählen:



Direktzugriff auf COM-Funktion

Im Weiteren kann dann direkt auf die COM-Funktionen zugegriffen werden. Im Unterschied zu der Vorversion von Visual Basic (VB 6) werden die Bibliotheken jetzt über Namespaces organisiert:

Statt

```
Dim boman as new AddBOManager
```

Schreiben Sie jetzt

```
Dim boman as new addisonbasics.AddBOMangaer
```

Oder sie deklarieren am Anfang des Moduls:

```
imports addisonbasics
```

In dem Fall können Sie ohne den Namespacenamen auf die Klassen zugreifen.

Integration in C#

Die Integration in C# ist nahezu identisch der Integration in Visual Basic.Net.
Sie fügen über die gleiche Funktion die COM-Bibliotheken hinzu und erhalten C#-Namespaces, die Sie direkt innerhalb von C# verwenden können:

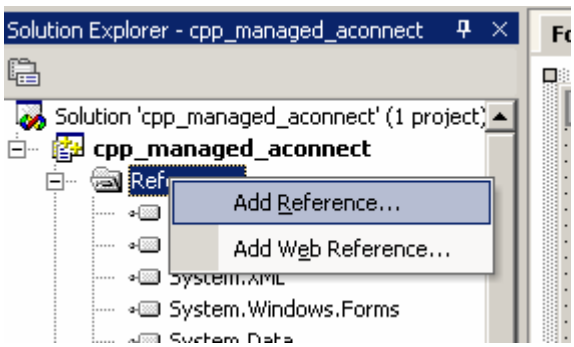
```
using System;
using addisonbasics;

namespace WindowsApplication1
{
	/// <summary>
	/// Summary description for Class1.
```

```
        /// </summary>
        public class Class1
        {
                public Class1()
                {
                }
                public void UseBOManager()
                {
                        AddBOManager boman = new AddBOManager
();

                        if (!boman.IsOpen ())
                        {
                                boman.Connect ();

                                ...

                                boman.Close ();
                        }

                }
        }
}
```

Integration in C++ (Managed Code)

Bei der Integration muss man unterscheiden, ob eine Einbindung auf Basis C++ mit Managed Code, mit Hilfe der Importfunktion oder mit Hilfe eines Wrappers (Dispatch-Interface) geschehen soll.

Managed Code:
Die Einbindung ähnelt dem Zugriff innerhalb von C# oder VB, der Referenzendialog wird allerdings über den Solutionexplorer (rechte Maustaste) aufgerufen:



Der Dialog ist dann der gleiche wie in VB (siehe Abschnitt Integration in C#).

Die Syntax des Zugriffs ist sprachabhängig dann wieder etwas verschieden:

```
#pragma once


namespace cpp_managed_aconnect
{
    using namespace System;
    using namespace System::ComponentModel;
```

```
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    using namespace Windows::Forms;

    using namespace Interop::addisonbasics;

    …

    private: System::Void button1_Click(System::Object *,System::EventArgs*e)
                  {
                        AddBOManagerClass * boman = new AddBOManagerClass ();

                        boman->Connect ();
                        if (boman->IsOpen)
                        {
                              AddMandant * pMandant = boman->get_GetMandant (1);
                              if (pMandant)
                              {
                                    MessageBox::Show (pMandant->m_Name);
                              }
                              boman->Close();
                        }
                  }

    };
}
```

Auf eine detaillierte Beschreibung soll hier verzichtet werden, der COM Zugriff ist mehr oder weniger ausführlich in der Literatur zum Thema Framework.NET beschrieben und soll hier nicht im Einzelnen erläutert werden. Praktikabler ist sicherlich, C++ für unmanaged und C# für managed Code zu verwendet, da hier viele Ungereimtheiten wie beispielsweise __gc Spezifier usw. nicht existieren, daher die C++-managed-Code-Einbindung nur exemplarisch.

Weiterführende Informationen

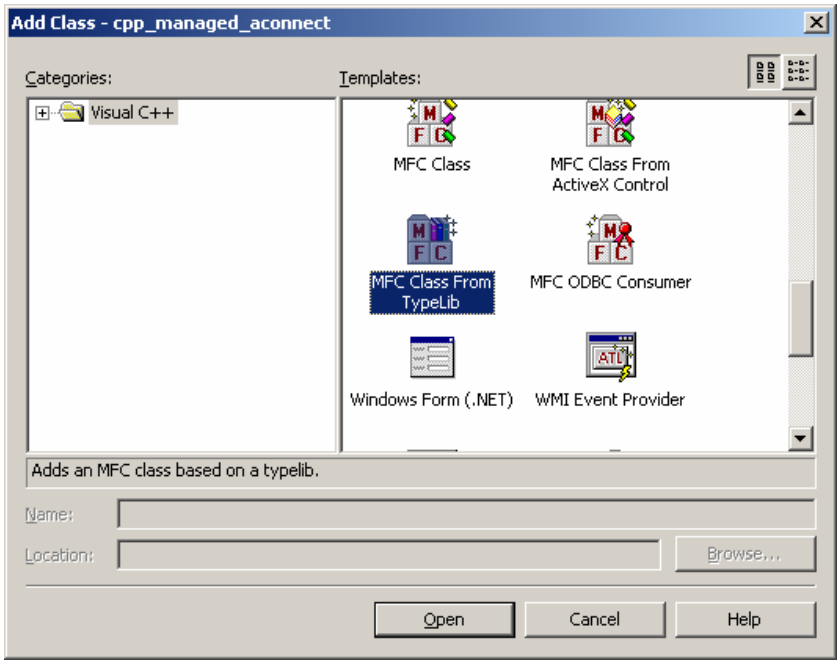
Weiterführende Informationen finden Sie in „.Net Framework“ in dem Buch „Programming with the .Net Framework“ innerhalb der MSDN (auch http://msdn.microsoft.com/) in dem Abschnitt „Exposing COM Components to the .NET Framework“.

Genauer Link:

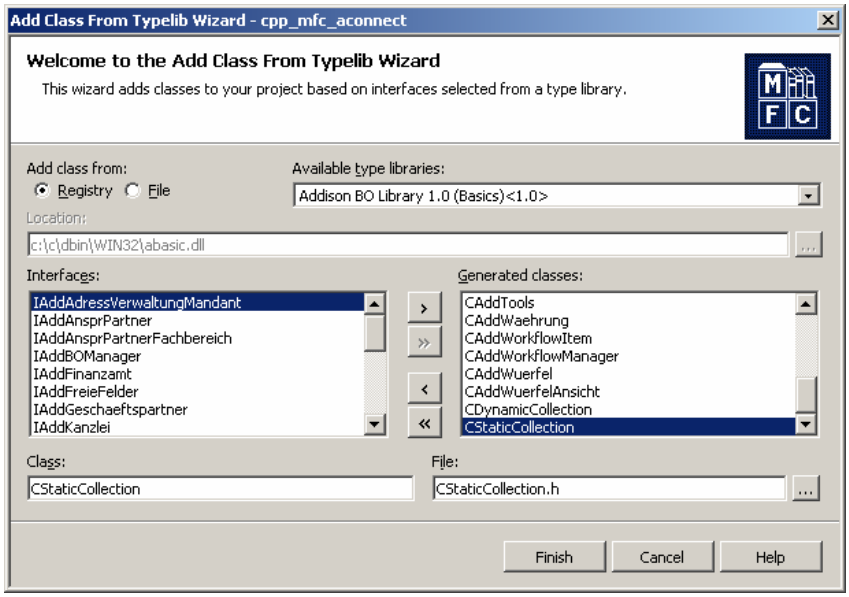
```
http://msdn.microsoft.com/library/default.asp?url=/library/enus/cpguide/html/cpconexposingcomcomponentstonetframework.asp
```

Integration in MFC Applikationen:

Das Hinzufügen zu MFC-Applikationen ohne Managed Code ist entsprechend der Version für Visual C++ 6.0, nur ist die Menüführung eine andere. Entsprechend gibt es auch die weiteren angesprochenen C++-Zugriffsmöglichkeiten auf COM weiterhin.



Sie finden Generator für die MFC Wrapper-Class in dem Add Class-Dialog: (Solution Explorer → Rechte Maustaste → Add → Add Class).
Dann erscheint der folgende Dialog:



Mit diesem Dialog können dann die passenden Klassen generiert werden.

Anschließend kann der Code verwendet werden (siehe auch Abschnitt über C++ 6.0).

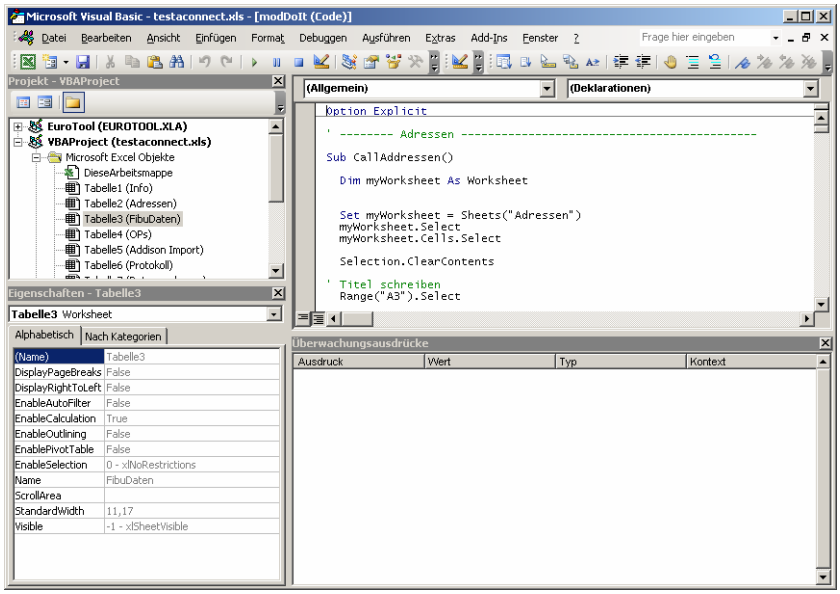
```
#include "CAddBOManager.h"
```

```
void TestFunktion()
{
CAddBOManager aBOMan;
aBOMan.CreateDispatch("abasic.AddBOManager");
aBOMan.Connect ();
if (aBOMan.get_IsOpen()!= 0L)
{
       ...
       aBOMan.Close ();
}
}
```

3.4. Visual Basic für Applikationen

Word und Excel

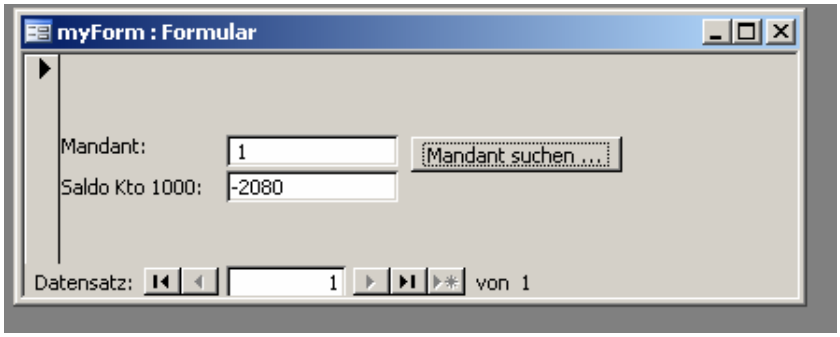
Die Integration in Word und Excel ist recht einfach und entspricht im Wesentlichen der Einbindung in Visual Basic. Über den Menüpunkt „Visual Basic Editor" (Alt+F11, Extras → Makro → Visual-Basic-Editor) gelangen Sie in die Makroentwicklungsumgebung von Word bzw. Excel.



Der Menüpunkt zum Eintragen ist im Menü „Extras → Verweise" zu finden und entspricht dem Dialog in Visual Basic.

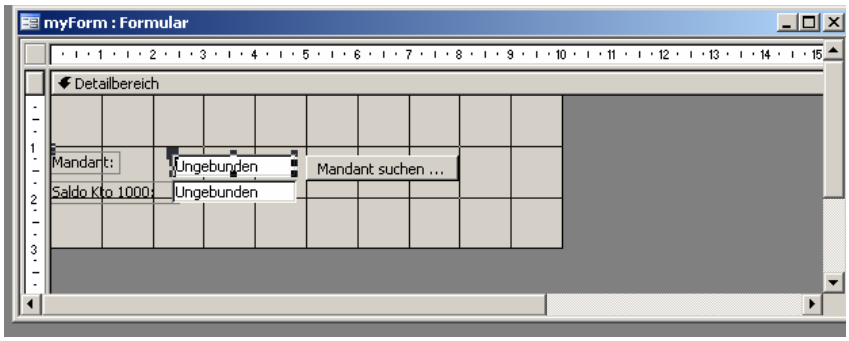
Microsoft Access

In den aktuellen Versionen von Access ist die Einbindung ähnlich, da auch hier jetzt Visual Basic für Applikationen integriert ist.
Sie referenzieren also auch hier den Zugriff innerhalb des Visual Basic-Fensters.
Beim Integrieren der Daten in ein Formular ist der Zugriff etwas anders als beim Einbinden einer Datentabelle, man muss sich mit den Möglichkeiten beschäftigen, das Formular über Events zu kontrollieren und über Visual Basic-Befehle das Formular füllen.



Anbindung Access-Formular

Um ein solches Formular anzubinden, geht man folgendermaßen vor:



Man legt die Formularfelder ungebunden an und programmiert innerhalb der VBA-Umgebung mittels Events. Beispielsweise das Anbinden des Buttons erfolgt mit folgendem Codefragment (die Edit-Felder heissen mandNr und Saldo1000):

```
Option Compare Database
Option Explicit

Private Sub Befehl0_Click()
  Dim myDlg As New MndSuchCtrl
  Dim l As Long
  l = myDlg.MndNummer(0)

  mandNr.Value = Str(l)

  Dim dlgExt As New dlgExtension
  Dim man As AddMandant
  Set man = dlgExt.GetMandant(l)

  If Not man Is Nothing Then

    Dim myFibu As New AddFibuProjectImp
    myFibu.AttachMandantenNr l

    If Not myFibu.GetLastWJ Is Nothing Then

      Dim myKonto As RwFibuKonto

      Set myKonto = myFibu.GetLastWJ.m_Sachkonten.SearchFibuKonto("1000")
      If myKonto Is Nothing Then
        Saldo1000.Value = "Konto nicht gefunden."
      Else
        Saldo1000.Value = myKonto.m_Saldo(15)
```

```
      End If
    Else
      Saldo1000.Value = "WJ nicht gefunden."
    End If
  Else
    Saldo1000.Value = "Mandant nicht gefunden."
  End If


  Set myDlg = Nothing

End Sub

Private Sub Form_Load()
  Dim dlgExt As New dlgExtension
  dlgExt.DoOpen

End Sub

Der Zugriff auf den BOManager ist innerhalb folgender Klassenfunktionen
  (Klasse dlgExtension) realisiert (Anwendung im Event Form_Load):

Public Function DoOpen() As Integer
    DoOpen = -1
    If m_BOManager Is Nothing Then
        Set m_BOManager = New AddBOManager
        If m_BOManager.IsOpen = 0 Then
          hasOpened = True
          m_BOManager.Connect
        End If
        If m_BOManager.IsOpen <> 0 Then
            DoOpen = 0
        End If
    Else
      DoOpen = 0
    End If
End Function

Function DoClose() As Integer
    If Not m_BOManager Is Nothing Then
        If hasOpened Then
Rem          m_BOManager.Close
        End If
        Set m_BOManager = Nothing
    End If
End Function

Function verifyOpen() As Integer
    verifyOpen = -1
    If Not m_BOManager Is Nothing Then
        If DoClose = 0 Then
            verifyOpen = 0
        End If
    End If
End Function

Private Sub Class_Initialize()
  hasOpened = False

'    MsgBox "Initialize"
'    Set m_BOManager = Nothing
End Sub

Private Sub Class_Terminate()
'    MsgBox "Terminate"
    If Not m_BOManager Is Nothing Then
        DoClose
    End If
```

```
End Sub

Private Function verifyNr(nr As Long) As Boolean
  If DoOpen = 0 Then
    Dim myMandant As AddMandant

    Set myMandant = m_BOManager.GetMandant(nr)
    If Not myMandant Is Nothing Then
      verifyNr = True
    Else
      verifyNr = False
    End If
  End If
End Function


Function GetMandant(nr As Long) As AddMandant
  If DoOpen = 0 Then
    Set GetMandant = m_BOManager.GetMandant(nr)
  Else
    Set GetMandant = Nothing
  End If
End Function
```

3.5. Andere Umgebungen

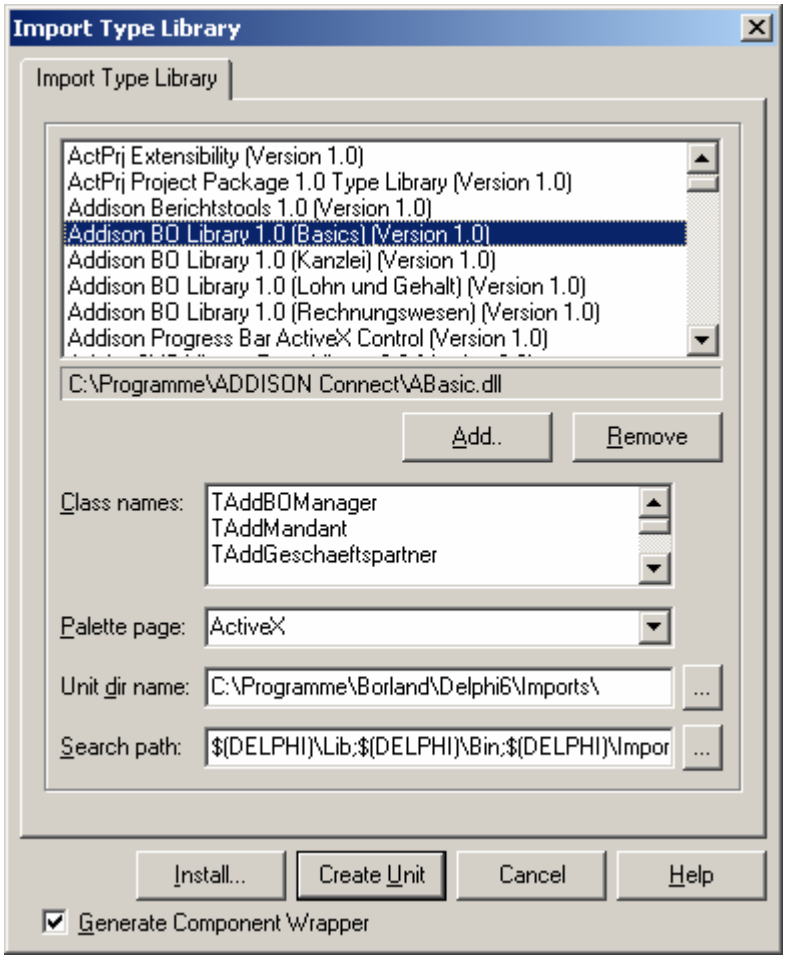
Java

Eine direkte Integration ist nicht möglich. Man muss zu diesem Zweck entweder ein COM-Wrapper-Werkzeug (bspw.) verwenden oder alternativ sich ein JNI-Interface schreiben.
Siehe auch:
http://www.ezjcom.com
http://www.ablon.de
http://www.cs.wustl.edu/~pjain/java/ace/ACE.html

Eine verbesserte Zugriffsfunktion ist geplant.

Delphi

Benutzen Sie den Menüpunkt „Project → Import Type Library ...“. In diesem Dialog finden Sie die Bibliothek und es wird eine Wrapper-Modul/Unit erzeugt, mittels dem Sie auf die Funktionen von ADDISON Connect zugreifen können.



Es wird dann bspw. ein Wrapper mit Namen addisonbasics_TLB erzeugt.

Die Anlage der Objekte erfolgt auf Basis der Units addisonbasics_TLB, ActiveX und ComObj, die Sie im Zweifelsfall hinzuziehen:

Im Beispiel sehen Sie in ein „Form“ integrierten ADDISON Connect-Code:
Die ADDISON Connect-Befehle sind hervorgehoben.

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, addisonbasics_TLB, Buttons, ComObj, ActiveX;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;

    …
    procedure TForm1.Button1Click(Sender: TObject);
    var
```

```
  boman : IAddBOManager;
  isOpen : Integer;
begin
  boman := CreateComObject (CLASS_AddBOManager) as IAddBOManager;
  boman.Connect ();
  isOpen :=  boman.IsOpen;
  if (isOpen <> 0) then
  begin
    Memo1.Text := 'BOManager geöffnet.';
  end
  else
  begin
    Memo1.Text := 'BOManager nicht geöffnet.';
  end;

end;

procedure TForm1.BitBtn2Click(Sender: TObject);
var
  boman : IAddBOManager;
  mandant : IAddMandant;
  isOpen : Integer;
begin
  boman := CreateComObject (CLASS_AddBOManager) as IAddBOManager;
  isOpen := boman.IsOpen;
  if isOpen <> 0 then
  begin
    mandant := boman.GetMandant[1];
    …

    if mandant <> Null then
      Memo1.Text := 'Mandant 1\n';
  end
  else
    ShowMessage('BOManager ist nicht geöffnet.');

end;

end.
```

Der Zugriff erfolgt also immer nach der Mimik: Deklarieren des Interfaces als Variable und Anlage über die GUID mit der Methode CreateComObject oder über eine Funktion des COM-Objekts (z.B. Methode GetMandant, die ein Mandantenobjekt anlegt).
Mit diesen Änderungen kann man den Beispielcode auf testaconnect.xls direkt übertragen.

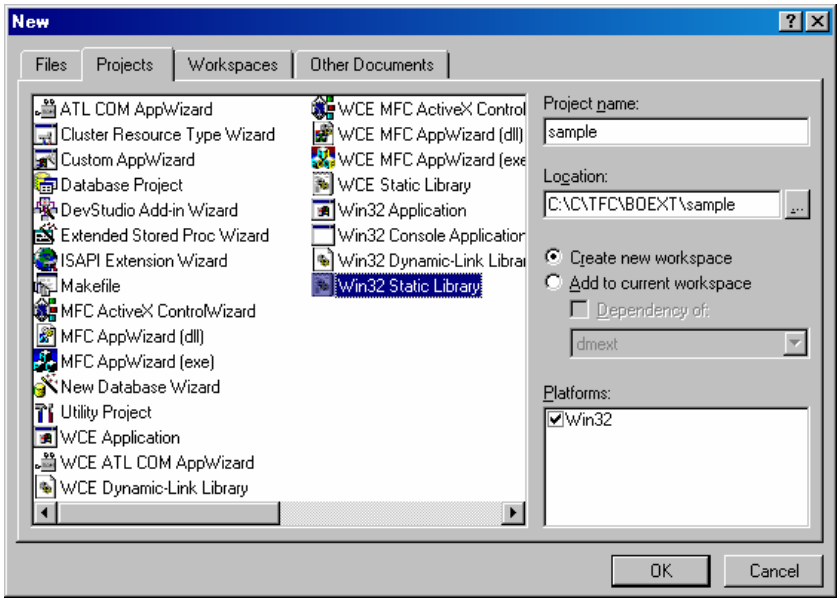
Sie können bei Bedarf auch eine Beispielapplikation oder direkte Unterstützung durch ADDISON erhalten.

4. Erstellung einer BO-Erweiterungs-DLL

4.1. Anlage der DLL

Erweiterungs-DLL

Sie brauchen, um eine Erweiterungs-DLL anzulegen, mindestens eine C++-Entwicklungsumgebung und den Header boext.h. Der Header sollte über einen Includepfad auf das ausgelieferte ZMiS-Verzeichnis zmis\include eingebunden werden.



Warum C statt C++ oder COM?

Die Schnittstellen sind deswegen in C gehalten, um eine einfach zu gewährleistende Binärkompatibilität bei hoher Performance zu gewährleisten.

In den Übergabestrukturen werden Versionsnummern hinterlegt, die intern weiter unterstützt werden, d.h., Sie können eine Erweiterungs-DLL, die mit einer alten Version des boext.h geschrieben ist, weiter verwenden, ohne sie neu zu kompilieren. Vergleichbare Mechanismen gibt es auch in COM und C++, aber sie sind technisch recht kompliziert und daher störungsanfällig.

Wenn Sie in C/C++ nicht über große Erfahrung verfügen, nehmen Sie entweder zur Prüfung ihres Codes oder zur Unterstützung mit uns Kontakt auf und halten Sie sich genau an diese Beschreibung.

4.2. Implementieren der Funktionen

Funktionen

Die möglichen Funktionen sind innerhalb der Datei "boext.h" deklariert. Eine Schnittstelle besteht immer aus einer Sammlung von Funktionen, die innerhalb der DLL implementiert sind und einer Struktur, innerhalb derer die Funktionen an ZMiS übergeben werden.

Das Laden der DLL erfolgt beim Starten der DLL aus dem boext-Verzeichnis in ZMiS. Die Steuerung der zu ladenen DLLs erfolgt über die boext.ini, in der das Laden von Erweiterungs-DLLs unterdrückt werden kann.

```
[IgnoreExtension]
adextprj.dll=1
```

Durch diesen Eintrag wird das Laden der DLL adextprj.dll unterdrückt. Im ZMiS-Setup wird ein Programm installiert, mit dem die boext.ini verwaltet wird.
Nach dem Laden der DLL versucht ZMiS die exportierten Zugriffsfunktionen zu finden:

```
// adextprj.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include <afxdllx.h>
#include "..\inc\boext.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


static AFX_EXTENSION_MODULE AdextprjDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    UNREFERENCED_PARAMETER(lpReserved);

    if (dwReason == DLL_PROCESS_ATTACH)
    {
           TRACE0("ADEXTPRJ.DLL Initializing!\n");

           if (!AfxInitExtensionModule(AdextprjDLL, hInstance))
                  return 0;

           new CDynLinkLibrary(AdextprjDLL);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
           TRACE0("ADEXTPRJ.DLL Terminating!\n");
           AfxTermExtensionModule(AdextprjDLL);
    }
    return 1;
```

```
}



extern int AddKontoChanged ( unsigned long mandnr, const char * kontoNummer, int
tag ,int monat, int jahr );
extern int AddShowBeleg ( unsigned long mandnr, const char * docId );
extern int AddCanShowBeleg (const char * docId );
extern int AddInit () ;
extern int AddDeInit () ;


extern "C"
{
__declspec(dllexport) int InitBOExtension (const char * cuser, const char * cpwd)
{
    AddInit ();
    return 0;
}

/**
 * Deinitialisieren der Extension
 *
 * @return
 */
__declspec(dllexport)int DeInitBOExtension ()
{
    AddDeInit () ;
    return 0;
}

/**
 * Holen des Erweiterungstools
 *
 * @param nummer
 * @return
 */
__declspec(dllexport) int GetExtensionInterface ( int nummer,  void *& pExtension)
{
    if (nummer == EXTENSION_EVENT)
    {
           pExtension = new BOEventInterface ();
           BOEventInterface * pInterface = (BOEventInterface *)pExtension;
           pInterface->m_pKontoChangedHandler = AddKontoChanged;
           pInterface->m_pShowBeleg = AddShowBeleg;
           pInterface->m_pCanShowBeleg = AddCanShowBeleg;
           return 0;
    }
    return -1;
}

/**
 * Löschen der angelegten Ressourcen.
 * Man muß diese Sachen hier selbst löschen, um
 * Probleme verschiedenen Speicherverwaltungssystem zu umgehen
 *
 * @param nummer
 * @return
 */

__declspec(dllexport) int  DeleteExtensionInterface ( int nummer,     void *
pExtension )
{
    if (nummer == EXTENSION_EVENT)
    {
           delete (BOEventInterface *)pExtension;
    }
```

```
    return 0;
}


}
```

Codebeispiel

In diesem Codebeispiel ist der Export eines solchen Zugriffsinterfaces beispielhaft implementiert:

```
__declspec(dllexport)int InitBOExtension (const char * cuser, const char * cpwd)
__declspec(dllexport)int DeInitBOExtension ()
__declspec(dllexport)int GetExtensionInterface ( int nummer,   void *& pExtension)
```

Diese 3 Funktionen sind entscheidend: InitBOExtension wird nach dem Laden und DeInitBOExtension wird vor dem Entladen der DLL aufgerufen. Über die Methode GetExtensioninterface wird die mit Funktionspointern gefüllte Struktur nach außen gegeben.
Die Strukturen sind definiert innerhalb der Datei boext.h.

Auflistung der Interfaces:

- DocManInterface: Anbindung Dokumentenmanager
- AddWuerfelExtensionInterface: Erweiterung Data Warehouse
- BOEventInterface: Verknüpfung Belegaufruf, z.B. um Rechnung aus Kontoblatt heraus aufzurufen
- ReweImportInterface: Erweiterung FIbu-Importschnittstelle
- GroupwareSyncInterface: Anbindung Groupware (z.B. Outlook)
- FremddatenUebernahmeInterface: Übernahme Fremddaten, Konvertierungsprogramme zum Zugriff bei der Datenübernahme

Beispiel

Im Verzeichnis „c:\programme\ADDISON Connect\samples" gibt es das komplexere Beispiel „dtvimpex", in dem beispielhaft eine Importschnittstelle implementiert ist.
Sie können bei ADDISON Software und Service GmbH direkt weitere Beispiele für die Implementierung der Schnittstellen und Unterstützung bei der Entwicklung erhalten.

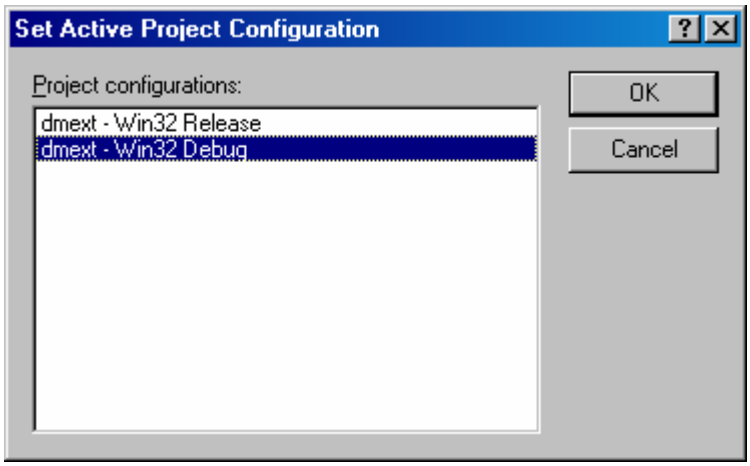
4.3. Debuggen der Funktionen

Voraussetzungen

Um die Funktionen zu Debuggen, müssen Sie ZMiS im Debugger laden oder sich in den Prozess zu attachen. Es ist zu empfehlen, dass Sie das ZMiS im Debugger laden, das Projekt entsprechend einstellen, dass Sie ihrer Debug-DLL entsprechend laden und dann in den Funktionen einen Breakpoint setzen.
Voraussetzung ist, dass Sie die DLL mit Debuginformationen

kompiliert haben.

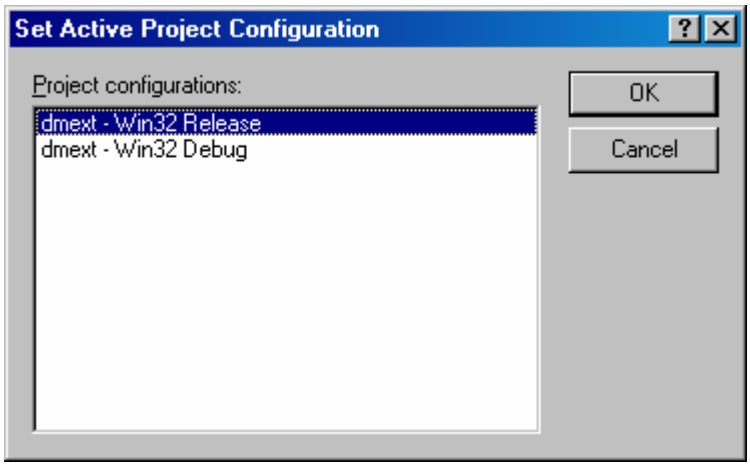
Menü Build: Set Active Configuration ... : Auswahl Debug



4.4. Auslieferung

Kompilierung

Wenn Sie die erstellte DLL ausliefern wollen, kompilieren Sie die DLL im Releasemodus:



Andernfalls können Sie nicht auf die Debug-DLLs (z.B. Debug-C-Runtime-DLL) zugreifen.

5. Installation und Betrieb ohne ZMiS-Installation

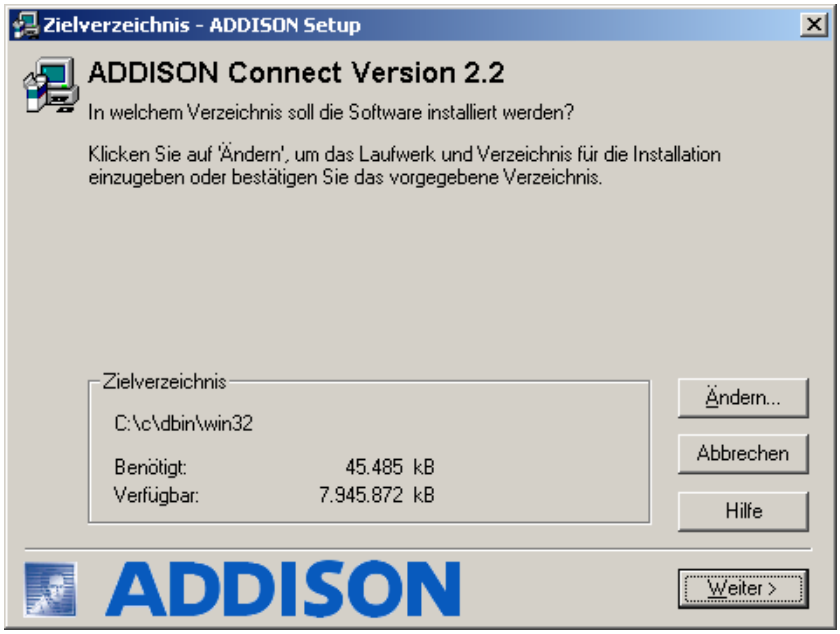
5.1. Anwendungsfall

Anwendungsfall

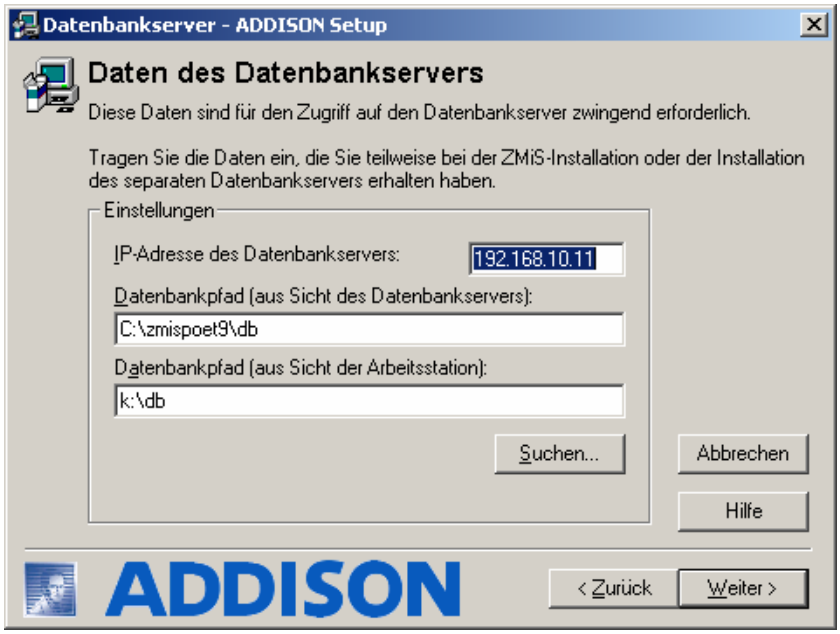
Wenn die Anwendung auf einem getrennten Rechner ohne ZMiS-Installation laufen soll, kann man die aconnect-Komponenten auch ohne ZMiS installieren. In dem Fall muss ein Netzwerkzugriff auf den POET-Server über TCP/IP verfügbar sein.
Das kann zum Beispiel notwendig sein, wenn Sie eine Branchensoftware auf einem Rechner installieren, der keinen Zugriff auf die Finanzbuchhaltung haben soll und trotzdem eine Verbindung zwischen der Software und ZMiS bestehen soll.

Vorgehensweise

Sie können über das Ausführen der aconnect.exe (wird im ZMiS-Verzeichnis installiert) Aconnect auch manuell installieren:



Der Setup-Dialog dient zur Auswahl des Verzeichnisses, in dem ADDISON Connect installiert werden soll. Standardmäßig wird aconnect unter c:\programme\ADDISON Connect installiert.



Nach Betätigen von „Weiter“ gelangen Sie in den Einstellungsdialog:

IP-Adresse

Datenbankpfad (aus Sicht des Datenbankservers)

Der Pfad zum Zugriff auf die Datenbank wird an den Server übermittelt, er muss den Pfad der Datenbank auf dem Server enthalten.

Datenbankpfad (aus Sicht der Arbeitsstation)

Beispiel

Angenommen, auf dem Server ist das ZMiS installiert unter c:\addison\zmis, und c:\addison ist vom Server aus freigegeben und auf der Arbeitsstation als Laufwerk H: gemapt.

Datenbankpfad Server: C:\addison\zmis\db
Datenbankpfad Arbeitsstation: H:\zmis\db

Ende der Installation

Nach Beendigung der Installation ist ADDISON Connect installiert und betriebsbereit.