Martin Rabe

Entwicklung eines Excel 2003 Add-In mit Microsoft Visual Studio 2008 zur Integration verschiedener Datenbanksysteme.

Mit Funktionsbeispiel für die Schloz Wöllenstein GmbH & Co. KG

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Informationstechnik & Elektrotechnik

Mittweida, 2010

Martin Rabe

Entwicklung eines Excel 2003 Add-In mit Microsoft Visual Studio 2008 zur Integration verschiedener Datenbanksysteme.

Mit Funktionsbeispiel für die Schloz Wöllenstein GmbH & Co. KG

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Informationstechnik & Elektrotechnik

Mittweida, 2010

Erstprüfer: Prof. Dr. rer. nat. Sergej Alekseev

Zweitprüfer: Dipl.-Ing. Lutz Lörper

Vorgelegte Arbeit wurde verteidigt am 10.05.2010

Bibliografische Beschreibung:

Rabe, Martin:
Entwicklung eines Excel 2003 Add-In mit Microsoft Visual Studio 2008 zur Integration verschiedener Datenbanksysteme, Mittweida, Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik, Diplomarbeit, 2010

Referat:
Das Ziel der Arbeit ist die Entwicklung eines Excel 2003 Add-In als kompaktes Integrationstool für verschiedene Datenbanksysteme. Das zu entwickelnde Add-In kann die meistgebrauchten Datenbankensysteme integrieren und bietet dem Nutzer eine einfache Möglichkeit, Daten mit administrierbaren Excelfunktionen oder selbsterstellten SQL-Abfragen auszulesen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CLR	Common Language Runtime
CLSID	Class Identifier
COM	Component Object Model
DaPoXL	Datenpool für Excel
DLL	Dynamic Link Library
DTD	Document Type Definition
GAC	Global Assembly Cache
GUID	Global Unique Identifier
HTML	Hyper Text Markup Language
ODBC	Open Database Connectivity
OLE DB	Object Linking and Ebedding Database
PHP	Hypertext Preprocessor
ProgID	Programmatic Identifier
SDK	Software Development Kit
SQL	Structured Querry Language
URL	Uniform Resource Locator
VB.NET	Visual Basic .NET
VBA	Visual Basic for Application
VS2008	Visual Studio 2008
VSTO	Visual Studio Tools für Office
VSTOR	Visual Studio Tools für Office Runtime
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Diese Diplomarbeit wurde im Zeitraum von September 2009 bis März 2010 bei der Schloz Wöllenstein GmbH & Co. KG in der Geschäftsführung Chemnitz angefertigt.

Ich möchte mich an dieser Stelle für die außerordentliche Unterstützung bei Herrn Michael Reiß, dem kaufmännischen Leiter, bei Herrn Dipl.-Ing. Lutz Lörper, dem Leiter der EDV-Abteilung, und Herrn Dipl.-Ing. André Uhlig bedanken, die mir trotz ihrer vollen Terminkalender bei Fragen mit Rat und Tat zur Seiten standen.

1. Einleitung

Die Schloz Wöllenstein GmbH & Co. KG bietet als mittelständisches Unternehmen seinen Kunden eine Vielzahl an Dienstleistungen und Produkten rund um das Thema Auto und Lastkraftwagen an. Durch Kundenbetreuung, Reparaturen, Leasing, sowie An- und Verkauf ist es selbstverständlich, dass ein hohes Datenvolumen aufkommt. Infolge dessen müssen diese Daten in den verschiedenen Abteilungen, wie beispielsweise Disposition oder Buchhaltung, zu pflegen sein, was durch die Verwendung diverser Datenbanken gewährleistet wird.

Für die Datenpflege der verschiedenen Datenbanken nutzen die Mitarbeiter mehrere umfangreiche Programme. Um jedoch kurzerhand Informationen für flexible Auswertungen in Excel zusammenzustellen, sind diese ungeeignet. Darum sollte eine übersichtliche Anwendung entwickelt werden, die sich direkt in Excel integriert und eine Brücke zu den verschiedenen Datenbanken schlagen kann. Darüber hinaus soll dem Anwender eine grafische Oberfläche geboten werden, die auf verschiedene Arten eine Datenbankabfrage vollziehen kann.

Um jedoch ein Excel Add-In auf verschiedenen Computern benutzen du können, bedarf es einigen Vorkehrungen. Aus diesem Grund beinhaltet diese Diplomarbeit grundlegende Details für die Entwicklung von Excel Add-Ins, sowie einen Einblick in die betreffenden Add-In Arten. Des Weiteren werden die Systemspezifikationen erläutert, die als Voraussetzung für die Ausführung von Excel Add-ins notwendig sind. Folgend werden Struktur, angewandte Zusammenhänge und Funktionsweise des zu entwickelnden Add-In beschrieben, bevor abschließend die Ergebnisse zusammengefasst und Ausblicke für die Weiterentwicklung vorgestellt werden.

2. Excel 2003 Add-Ins mit Microsoft Visual Studio

2.1 Microsoft Visual Studio

Bei Programmierern und Entwicklern für Office- uns Windows-Anwendungen ist die Entwicklungsumgebung Microsoft Visual Studio recht weit verbreitet. Diese dient im Allgemeinen zur Entwicklung von Windowsprogrammen, kann aber ebenfalls für die Programmierung von Webseiten oder beispielsweise mobilen Endgeräten genutzt werden. Microsoft bietet damit Entwicklern die Möglichkeit, Anwendungen in den Programmiersprachen Visual Basic .Net, C, C++, und C-Sharp (C#) zu programmieren. Unterstützt werden ebenfalls HTML, PHP und XML.
In der aktuellen Visual Studio 2008 Version (im Folgenden VS2008 genannt) können Programmierer auf eine Vielzahl an Vorlagen zurückgreifen. So gibt es vorkonfigurierte Projekte für Microsoft Office Anwendungen. Für Excel 2003 Add-Ins bestehen die Voreinstellungen aus einem Add-In Projekt und einem Setup-Projekt. Durch das Verwenden dieser Vorlage werden diverse Abhängigkeiten zwischen den Projekten, wie beispielsweise der notwendige Registrierungseintrag für Excel bereitgestellt. Jedoch ist dies meistens nicht ausreichend, um das spätere Add-In im Excel ausführen zu lassen.
Am Computer des Programmierers, als auch des Nutzers, ist das Microsoft .NET-Framework und die Visual Studio Tools for Office Runtime unumgänglich.

2.1.1 Das .NET-Framework

Nach [5] ist das .NET-Framework (sprich dotnet) eine von Microsoft entwickelte Software-Plattform, die eine objektorientierte Laufzeitumgebung, die *Common Language Runtime* (CLR), und mehrere Klassenbibliotheken, den sogenannten *Application Programming Interface* (API), bereitstellt.
Die CLR ist eine sprachübergreifende Zwischenschicht. Sprachübergreifend deswegen, da sie Programme der verschiedenen Programmiersprachen interpretieren und in den für den Computer verständlichen Maschinencode umwandeln kann.
Die APIs stellen eine Sammlung an unterschiedlichen, im Programmcode wieder verwendbaren, Komponenten dar. Dies können beispielsweise Schaltflächen, Textfelder oder ein Timer sein, die somit die Funktionalität eines Programms umfassen.
Bibliotheken oder ausführbare Programme, die mit der .NET-Technologie entwickelt werden, gelten als verwaltete Programme (managed) und werden im Allgemeinen als Assemblys bezeichnet. Es werden unter anderem Arbeitsspeicher und Prozesse überwacht, um sicherzustellen, dass keine Speicherverluste auftreten und somit die Zuverlässigkeit der Anwendung erhöht wird. Alle weiteren Programme werden als nicht verwaltet (unmanaged) bezeichnet, wenn diese nicht auf Basis der .NET-Technologie arbeiten.

2.1.2 Visual Studio Tools for Office

Um Microsoft Office Applikationen erstellen zu können, benötigt ein Entwickler neben dem .NET-Framework zusätzlich die Visual Studio Tools for Office Runtime (VSTOR). Diese stellt ein Bindeglied dar, um das .NET-Framework in Microsoft Office Anwendungen zu integrieren und somit verwaltete Assemblys verwenden zu können. Dadurch kann beispielsweise Excel diverse Add-Ins einbinden, die in verschiedenen Programmiersprachen geschrieben wurden, um damit dessen Funktionalität zu erweitern.

2.2 Arten der Microsoft Excel 2003 Add-Ins

Excel, das am weitesten verbreitete Tabellenkalkulationsprogramm, bietet dessen Nutzern eine Vielzahl an Formel, Diagrammen, Funktionen und diversen Berechnungsmöglichkeiten. Dieser Umfang lässt sich mittels von Anwendern erstellten Makros, der integrierten Programmiersprache VBA oder präparierten Excel-Arbeitsmappen erweitern.
Microsoft Office Entwickler haben zudem die Möglichkeit, die Funktionalität Excels durch Add-Ins auszubauen. Dazu zählen das COM-Add-In und der Automatisierungsserver.

- COM-Add-In

  Das COM-Add-In dient in der Regel zur Erweiterung Excels um benutzerdefinierte Aufgaben. Das heißt, dass beispielsweise durch betätigen einer Schaltfläche oder auch durch Öffnen oder Schließen der Anwendung Ereignisse ausgelöst werden können, um eine Prozedur oder Reaktion hervorzurufen. Die direkte Interaktion mit Excel-Zellen ist einem COM-Add-In jedoch nicht möglich. Das Einbinden in Excel erfolgt meist als DLL.

- Automatisierungsserver

  Automatisierungsserver bzw. Automatisierungs-Add-Ins bauen auf COM-Add-Ins auf. Sie werden genutzt, um nutzererstellte Funktionen in Excel einzubinden. Dadurch kann der Anwender Interaktionen direkt über die Excel-Zelle auslösen. Auch Automatisierungs-Add-Ins werden als DLL eingebunden.

Damit das COM-Add-In und das Automatisierungs-Add-In ordnungsgemäß funktionieren können, müssen gewisse Voraussetzungen, wie Registrierungseinträge oder eine Rechtevergabe, abgehandelt werden. Diese Themen werden in Kapitel 3 näher erläutert.

3. Add-In Spezifikationen

3.1 Ladeverhalten von Add-Ins in Microsoft Excel 2003

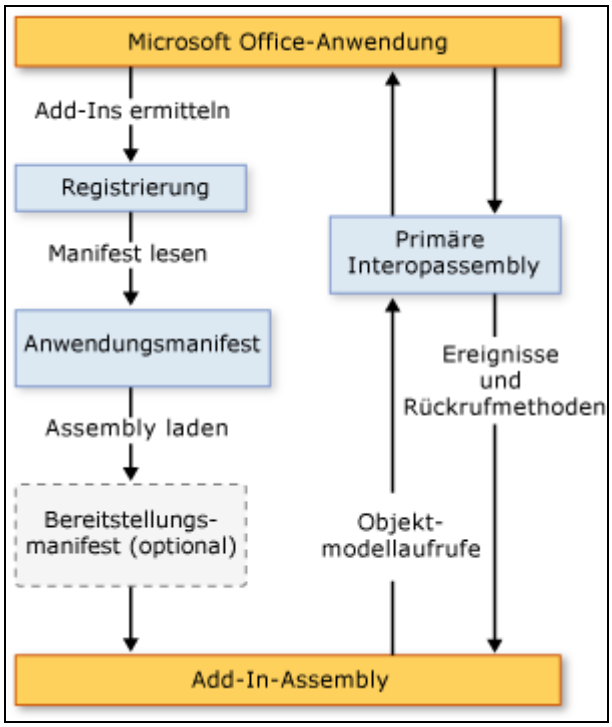


Abbildung 1 - Architektur eines Add-Ins für Microsoft Office 2003 [8]

Abbildung 1 zeigt das grundlegende Ladeverhalten einer Microsoft Office Anwendung für Add-Ins auf. Beim Start der Office Applikation, wie beispielsweise Excel, werden folgende Schritte abgehandelt:

1. Die Anwendung überprüft die Registrierungseinträge, die mithilfe von VSTO erstellt wurden, um das Add-In zu identifizieren.
2. Werden die Add-In spezifischen Einträge gefunden, lädt die Microsoft Office Anwendung die Datei „*VSTOEE.dll*“, welche wiederum die „*AddinLoader.dll*“-Datei lädt. Wie unter [6] beschrieben sind diese Dateien nicht verwaltete Komponenten der VSTO Laufzeit, die für den Ladeprozess der VSTO nötig sind.
3. Die „*AddinLoader.dll*“-Datei lädt das .NET-Framework und startet den verwalteten (managed) Teil der VSTO Laufzeit.

4. Die VSTO Laufzeit erstellt eine neue Anwendungsdomäne, die einen in sich abgeschlossenen Bereich darstellt, bei der bestimmte Anforderungen an das zu ladende Add-In gestellt werden. Das .NET-Framework prüft, ob das Add-In die Anforderungen erfüllt und lässt es daraufhin ausführen oder deaktiviert es.
5. Ist ein Bereitstellungsmanifest eines Add-In vorhanden, sucht die VSTO Laufzeit nach Assemblyupdates und aktualisiert das Add-In falls erforderlich.
6. Letztlich wird das Add-In von der VSTO Laufzeit in die zuvor angelegte Anwendungsdomäne geladen und dessen Start-Ereignis ausgelöst.
7. Die Kommunikation des geladenen Add-Ins mit der Microsoft Office-Anwendung erfolgt über die primären Interopassemblys. Nach [9] dienen diese Interopassemblys der Interaktion mit der jeweiligen Office-Anwendung und deren Komponenten (z.B. eine Excelzelle). Der Verweis auf die entsprechende Office-Applikation erfolgt automatisch durch die Verwendung der jeweiligen VSTO-Projektvorlage.

Durch die Beschreibung des Ladevorgangs wird ersichtlich, dass das Add-In neben dem .NET-Framework und der Visual Studio Tools for Office Laufzeitumgebung noch zwei weitere essentielle Voraussetzungen erfüllen muss, um ausgeführt werden zu können. Einerseits müssen die Registrierungseinträge vorhanden sein und andererseits eine Berechtigung, die als Codezugriffssicherheits-Richtlinie bezeichnet wird.

3.2 Die Windows-Registrierung

3.2.1 Grundaufbau der Windows-Registrierung

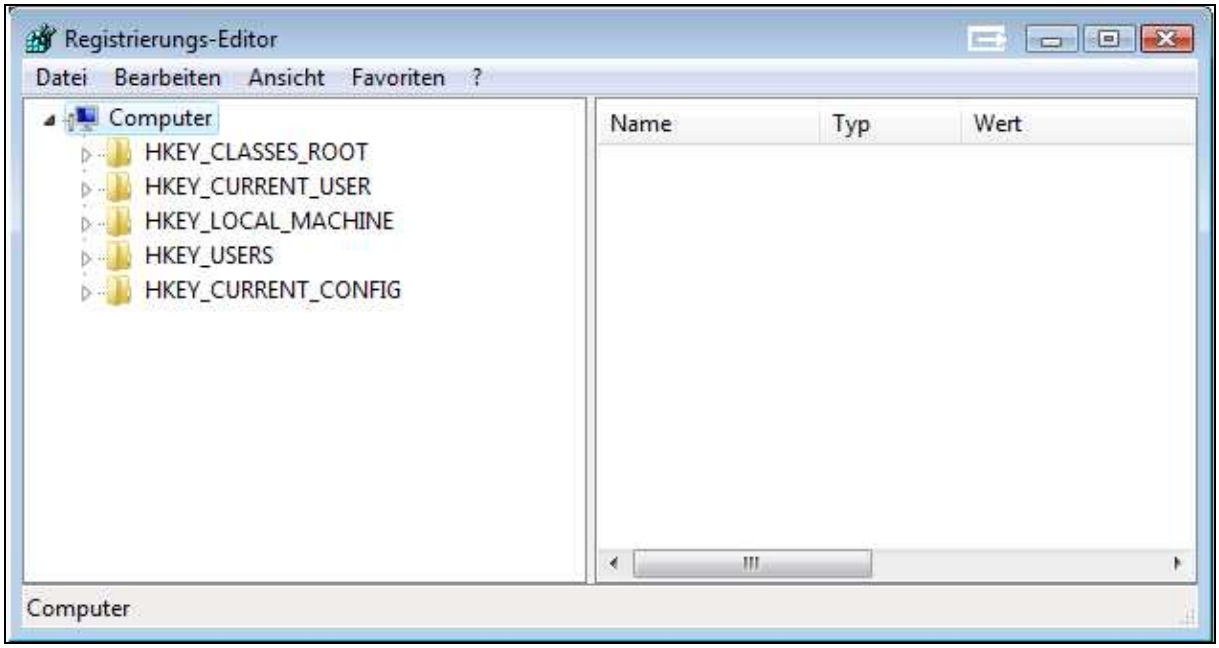


Abbildung 2 - Die Windowsregistrierung

Bei einem Windows-Betriebssystem werden Informationen zu Programmen, als auch zu Windows selbst, in einer Registrierungsdatenbank gespeichert (*Abbildung 2*). Diese wird im Allgemeinen als Registry bezeichnet, gliedert sich in einer Baumstruktur auf und wird in sogenannten Schlüsseln (Keys) abgelegt. Die Untergliederung aller Einträge erfolgt in fünf Hauptschlüssel, wie unter [10] beschrieben:

- HKEY_CLASSES_ROOT

  Unter diesem Key werden Informationen zu Dateien abgelegt, die einem bestimmten Programm zugeordnet sind. So wird beispielsweise per Doppelklick auf eine Datei das jeweils dafür vorgesehene Programm gestartet.

- HKEY_CURRENT_USER

  Im HKEY_CURRENT_USER-Key werden die Einstellungen des jeweils aktuell angemeldeten Benutzers gespeichert. Dieser wird von HKEY_USERS abgeleitet.

- HKEY_LOCAL_MACHINE

  Dieser Schlüssel beinhaltet sämtliche Hard- und Softwareinformationen eines Computers und gilt für alle Benutzer.

- HKEY_USERS

  In HKEY_USERS sind die benutzerdefinierten Einstellungen, wie Desktop- oder Netzwerkeinstellungen, aller Benutzer eines Computers gespeichert.

- HKEY_CURRENT_CONFIG

  Dieser Schlüssel verweist auf Einstellungen und Informationen zu diversen Peripheriegeräte, wie Drucker oder Scanner, des aktuell angemeldeten Benutzers.

3.2.2 Eindeutigkeit durch CLSID

Bei einem Windows-Betriebssystem werden verschiedene Programme und Installationen in der Windowsregistrierung durch sogenannte Class Identifier (CLSID) unterschieden. Diese stellen nach [11] eine spezielle Form der Global Unique Identifier (GUID) dar und beziehen sich auf eine 128-Bit-Ganzzahl. Eine CLSID ist ein eindeutiger Bezeichner in einem Betriebssystem und enthält 32 Hexadezimalziffern, die in einer bestimmten Zifferngruppe von 8-4-4-4-12 in geschweiften Klammern angeordnet sind.

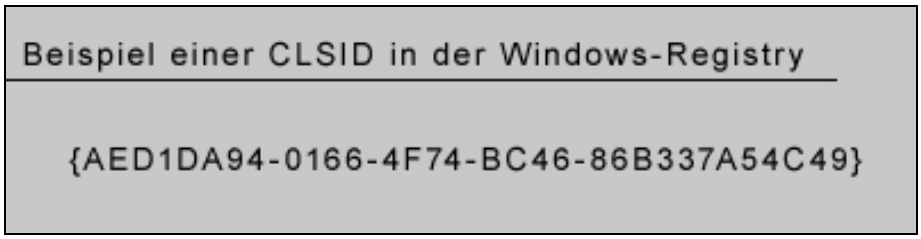


Abbildung 3 - Beispiel einer CLSID

Anhand *Abbildung 3* ist die spezielle Schreibweise einer CLSID bzw. GUID ersichtlich. Dabei geben der dritte Block und die ersten drei Zeichen des vierten Blocks den Typ, die Versionsnummer und den verwendeten Berechnungsalgorithmus an. Die restlichen Ziffern und Buchstaben einer CLSID erscheinen für den Betrachter zufällig, werden jedoch aus der Informationsmenge des jeweiligen Programms gebildet.

3.2.3 Registrierungseintrag eines Excel COM-Add-In

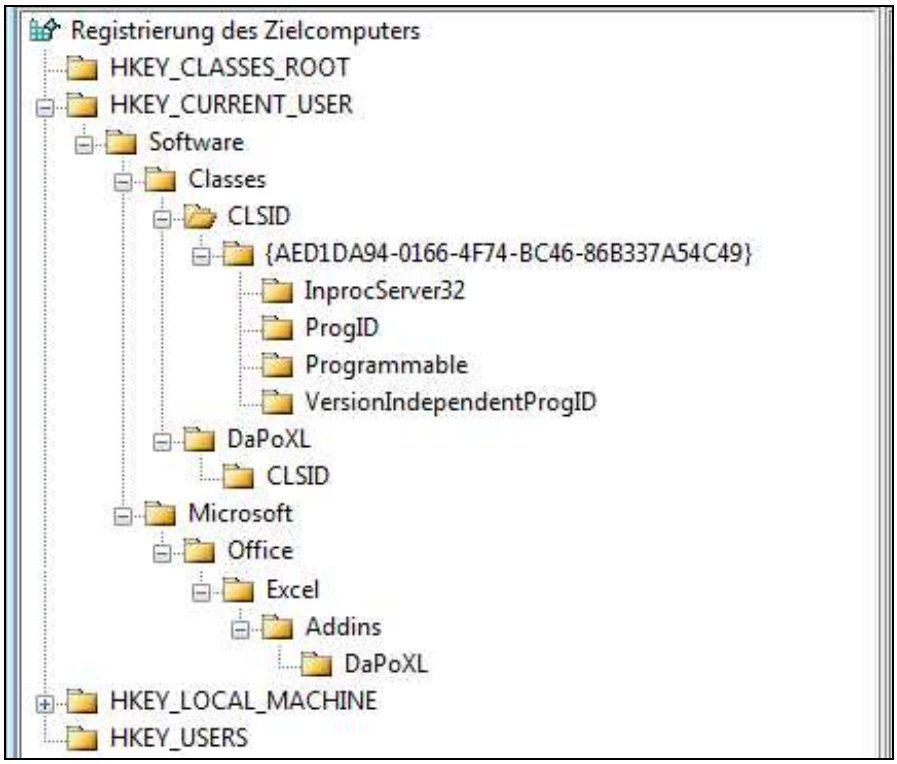


Abbildung 4 - Baumstruktur für das COM-Add-In „DaPoXL“

Abbildung 4 stellt ein Ausschnitt eines Add-In-Setupprojekts der Visual Studio 2008 Entwicklungsumgebung dar. Darauf wird die Baumstruktur der Registrierung des Add-Ins „DaPoXL“ abgebildet. Dieser Registrierungseintrag wird automatisch bei der Arbeit mit einer Excel 2003 Add-In-Vorlage im Setupprojekt angelegt. Der Entwickler kann vorhandene Schlüssel abändern, entfernen oder neue der Baumstruktur hinzufügen. Die verschiedenen Keys des Add-In werden durch die Entwicklungsumgebung zu den richtigen Hauptschlüsseln hinzugefügt. Die CLSID wird mir ihren verschiedenen Unterschlüssel ebenfalls automatisch durch Visual Studio generiert. Diese Unterschlüssel definieren das Add-In zusätzlich.

- InprocServer32

  Der Unterschlüssel „InprocServer32“ steht für In-Process-Server. Dieser ist meist eine DLL, die als Prozess in einer Anwendung ausgeführt wird. In diesem Fall ist das Add-In der In-Process-Server und Excel die Anwendungsdomäne bzw. Applikation, in der das Add-In ausgeführt werden kann. Der Schlüssel InprocServer32 enthält unter anderem Informationen des Installationspfades des Add-In und den Dateipfad der „*AddinLoader.dll*“.

- ProgID

  Die ProgID (Programmatic Identifier) repräsentiert den Programmnamen der zugrundeliegenden CLSID. Sie dient ebenfalls der Identifizierung der Anwendung, allerdings mit einer geringeren Genauigkeit im Gegensatz zur CLSID.

- Programmable und VersionIndependetProgID

  Diese beiden Unterschlüssel werden ebenfalls vom Visual Studio angelegt, sind jedoch zwangsläufig nicht von Bedeutung und könnten weggelassen werden.

Wird die Baumstruktur des „Classes“-Key weiterverfolgt, wird der Name des Add-In „DaPoXL“ zusätzlich eingetragen. Unter diesem Schlüssel wird nochmals die CLSID hinzugefügt. Dieser Unterschlüssel dient als weiterer Abgleich des Add-In Namens mit der dazugehörigen CLSID.
Der „Microsoft“-Schlüssel und dessen Unterschlüssel „Office“ stellen die Einträge der installierten Microsoft Office Anwendungen dar. In den Schlüsseln der Anwendungen existiert ein weiterer Unterschlüssel, die der Abhandlung von Add-Ins dient. Dieser Add-In-Key beinhaltet die Namen sämtlicher zum Einsatz kommender Add-Ins. Mithilfe der darin enthaltenen Namensschlüssel werden beispielsweise Ladeinformationen der Add-Ins verwaltet und der Office Anwendung bekannt gemacht.

3.2.4 Registrierungseintrag eines Excel Automatisierungs-Add-In

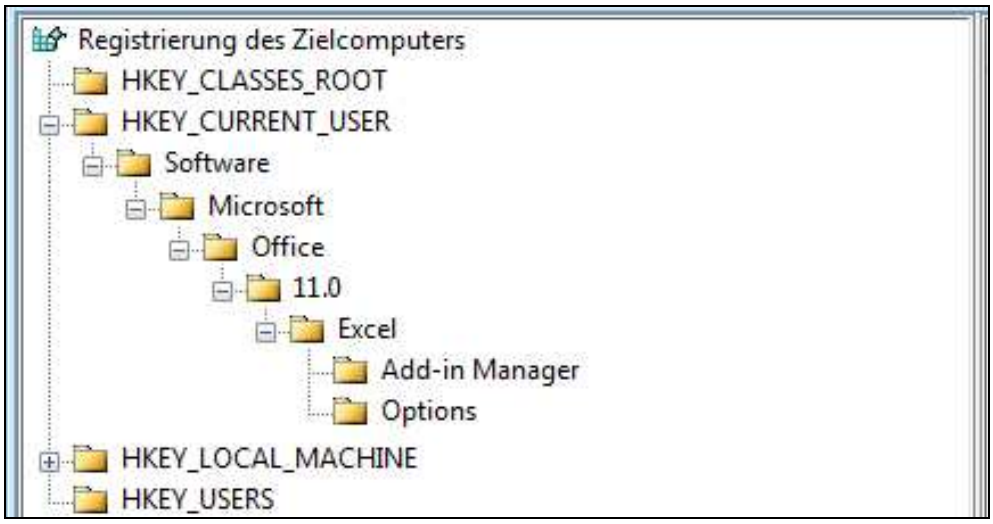


Abbildung 5 - Baumstruktur eines Automatisierungs-Add-In

Ein Automatisierungs-Add-In benötigt ebenfalls einen Registrierungseintrag. *Abbildung 5* stellt einen Ausschnitt des Setupprojektes der VS 2008 Entwicklungsumgebung dar. Im Vergleich zu einem Registrierungseintrag eines COM-Add-In werden die Schlüssel des Automatisierungs-Add-In nicht vom Visual Studio eingefügt, da dafür keine Vorlage existiert. Der Entwickler muss die benötigten Einträge per Hand und in einen separaten Unterschlüssel eintragen, der für Automatisierungs-Add-Ins vorgesehen ist. Der Schlüssel für Microsoft Office ist zwar zu einem COM-Add-In identisch, jedoch wird der Eintrag explizit auf die zu verwendende Version der Office-Anwendung festgelegt. In diesem Fall steht der Unterschlüssel „11.0“ für Office 2003. Microsoft Office 2007 hätte nun beispielsweise die Versionsnummer 12. Unter den jeweiligen Versionseintrag befinden sich wiederum die diversen Office Anwendungen. Im Unterschlüssel „Excel“ müssen für das Automatisierungs-Add-In zwei weiter Schlüssel eingetragen werden. Der „Add-in Manager“-Key beinhaltet den Namen des Add-In und unter dem Schlüssel „Options“ wird das Startverhalten des Add-In angegeben. Ein Automatisierungs-Add-In kann im Gegensatz zu einem COM-Add-In auch spät gebunden sein. Das bedeutet, dass es in die laufende Anwendung integriert werden kann, ohne dass die Anwendung neugestartet werden muss.

Der Eintrag einer CLSID und deren Unterschlüssel ist an dieser Stelle im Visual Studio nicht notwendig. Diese werden durch eine spezielle Funktion in die Registrierung hinzugefügt (siehe *Abbildung 6*).

```
'COM-Registrierungsfunktion für den Eintrag in die Windows-Registry
<ComRegisterFunction()> _
Public Shared Sub RegisterClass(ByVal t As Type)
    Dim key As RegistryKey = _
    Registry.ClassesRoot.CreateSubKey("CLSID\{" & t.GUID.ToString().ToUpper() & "}")
    key.CreateSubKey("Programmable")
    key.SetValue("", "DaPoXL-Funktionen")
    key.CreateSubKey("InprocServer32\").SetValue _
                ("", GetFolderPath(SpecialFolder.System) & "\mscoree.dll")
    key = Registry.ClassesRoot.CreateSubKey("DaPoXL.Funktionen")
    key.SetValue("", "DaPoXL-Funktionen")
End Sub
```

Abbildung 6 - Registrierungsfunktion für Automatisierungs-Add-In [2]

Die Registrierungsfunktion wird während der Installation des Add-In angesprochen und konfiguriert die vorzunehmenden Einträge. Der In-Process-Server für dieses Automatisierungs-Add-In ist eine Bibliothek (DLL) namens „mscoree.dll“, die vom .NET-Framework für diesen Zweck bereitgestellt wird.

3.3 Codezugriffssicherheit im .NET-Framework

3.3.1 Was ist die Codezugriffssicherheit?

Als Codezugriffssicherheit wird ein Sicherheitsverfahren im .NET-Framework beizeichnet, das Computersysteme vor bösartigen, mobilen Code schützt, die geschützte Ausführung von Code unbekannter Herkunft ermöglicht und absichtliche oder unbeabsichtigte Verletzung der Sicherheit durch vertrauenswürdigen Code verhindert. [12]
Diese Codezugriffssicherheit gilt für den gesamten verwalteten Code, der in der CLR des .NET-Framework ausgeführt wird. Selbst falls die ausgeführte Anwendung keinerlei Bezug zur .NET-Technologie haben sollte, wird die Codezugriffssicherheit erzwungen.
In diesem Zusammenhang gesehen, muss für ein Excel 2003 Add-In ebenfalls das Codezugriffsicherheitsverfahren angewandt werden. Es existieren verschiedene Codegruppen und Vertrauensebenen, die unabhängig der Rechte des jeweiligen Nutzers für die verschiedenen Assemblys zum Einsatz kommen und diversen Sicherheitsrichtlinien unterliegen.

3.3.2 Sicherheitsrichtlinien, Codegruppen und Vertrauensebenen

Richtlinientyp	Festlegende Person / Code
Unternehmensrichtlinie	Administrator
Computerrichtlinie	Administrator
Benutzerrichtlinie	Administrator oder Benutzer
Anwendungsdomänenrichtlinie	Code des Anwendungsanbieters

Tabelle 1 - Sicherheitsrichtlinien

Das .NET-Framework stellt vier Sicherheitsrichtlinien (*Tabelle 1*) für Assemblys oder Anwendungsdomänen zur Verfügung, welche sich in Unternehmens-, Computer-, Benutzer- und Anwendungsdomänenrichtlinie unterteilen. Diese spiegeln eine Hierarchie wieder, wobei die Unternehmensrichtlinie die oberste Ebene, die Computerrichtlinie die zweite, die Benutzerrichtlinie die dritte und

Anwendungsdomänenrichtlinie die unterste Ebene bildet. Beim Auslesen der verschiedenen Berechtigungen durchläuft die CLR diese Hierarchie von der obersten bis hin zur untersten Ebene. Niedrigere Richtlinienebenen können dabei Berechtigungen, die von einer höheren Ebene vererbt wurden, nicht erweitern, diese jedoch weiter einschränken.

Abgesehen von der Anwendungsdomänenrichtlinie, werden die Richtlinientypen von Codegruppen dargestellt. Die Anwendungsdomänenrichtlinie kann nicht administrativ angepasst, dafür aber programmgesteuert festgelegt werden. Eine Codegruppe ist eine logische Gruppierung von Code mit einer angegebenen Bedingung für die Mitgliedschaft.
Tabelle 2 zeigt die Mitgliedschaftsbedingungen auf, in welche die Codegruppen unterteilt sind. Erfüllt eine Assembly oder Anwendung eine Mitgliedschaftsbedingung, so ist diese in der dazugehörigen Gruppe enthalten.

Mitgliedschaftsbedingungen	**Beschreibung der Bedingung**
Gesamter Code	Stellt eine Bedingung dar, die von sämtlichen Code erfüllt wird.
Anwendungsverzeichnis	Das Installationsverzeichnis der Anwendung
GAC (Global Assembly Cache)	Eine Bibliothek im Global Assembly Cache
Kryptografischer Hash	Übereinstimmender Hashwert von Installation zu Autorenvorgabe
Softwarehersteller	Der öffentliche Schlüssel einer gültigen Authentifizierungssignatur
Sitemitgliedschaft	Die Internetseite oder der FTP-Server von dem der Code stammt
Starker Name	Ein digitaler Schlüssel, der dem Code als Signatur zugrunde liegt
URL (Uniform Resource Locator)	Die explizite Adresse einschließlich des Dateinamens der Ressource
Zone	Die Zone aus der Code stammt (z. B. Intranet)

Tabelle 2 - Mitgliedschaftsbedingungen der Codegruppen

Anhand der Beweise der Assembly, die als Mitgliedschaftsbedingung angegeben werden, prüft die CLR diese auf Korrektheit. Sind diese Angaben zutreffend, wird der Assembly die zugewiesene Berechtigung erteilt.

Vertrauensebenen	Erklärung
Volle Vertrauenswürdigkeit (Full-Trust)	Ist diese Berechtigung gesetzt, ist dem Code die Ausführung sämtlicher Aktionen gewährt, die auch vom Benutzer ausgeführt werden können.
Teilweise vertrauenswürdig	Diese Ebene ist ein eingeschränkter Berechtigungssatz, mit dem nur bestimmte Berechtigungen gewährt werden.
Nicht vertrauenswürdig	Mit dieser Ebene werden keine Berechtigungen gesetzt und der Code wird nicht ausgeführt.

Tabelle 3 - Vertrauensebenen einer Assembly

Tabelle 3 zeigt die drei möglichen Vertrauensebenen, die einer Assembly zugeteilt werden können. Um eine Microsoft Office Assembly, wie ein Excel Add-In es ist, ausführen zu können, muss diesem Add-In volle Verrauenswürdigkeit eingeräumt werden. Anderen Falls würde das Add-In von der Common Language Runtime automatisch blockiert und deaktiviert werden.

3.3.3 .NET-Konfigurationstool

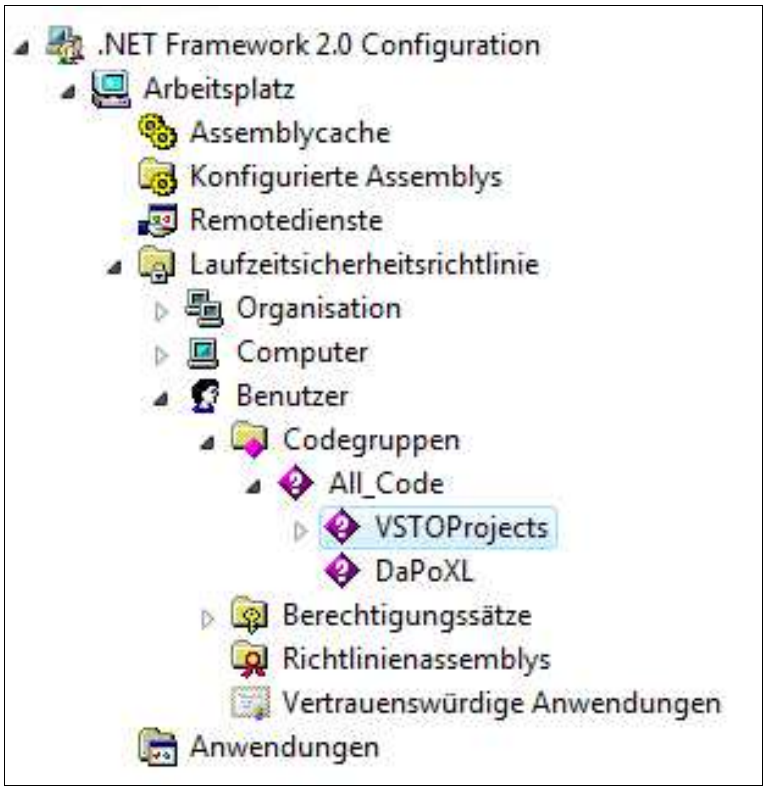


Abbildung 7 - Ausschnitt des .NET-Konfigurationstool

Um einen Einblick in die Struktur der Sicherheitsrichtlinien, Codegruppen, Vertrauensebenen und Berechtigungssätze erhalten zu können, wird ein Konfigurationstool von dem .NET-Framework Software Development Kit (SDK) bereitgestellt. Administratoren und Benutzer können im Konfigurationstool manuell die Einstellungen für verschiedene Anwendungen setzen oder bestehende abändern. In der Regel wird dies aber über Programmcode vollzogen. Auf *Abbildung 7* ist ein Ausschnitt dieses Konfigurationstools abgebildet, um die unter Punkt 3.3.1 erklärten Begriffe visuell zu unterstützen.

An der Benutzersicherheitsrichtlinie wird die Untergliederung der dazugehörenden Codegruppen ersichtlich. Die Codegruppe „All_Code“ gewährt dem gesamten Code absolute Vertrauenswürdigkeit und bildet den Stamm dieser Codegruppenstruktur. Darin enthalten sind zwei weitere Codegruppen. „VSTOProjects“ ist eine Codegruppe, die vom Visual Studio automatisch angelegt wird, sobald ein Entwickler sein erstelltes VSTO-Projekt im Visual Studio testet. Daraufhin wird die für das Projekt notwendige Berechtigung der vollen Vertrauenswürdigkeit, auf Basis der Mitgliedschaftsbedingung „URL“, erteilt, um die Office-Anwendung ausführen zu

dürfen. Die Codegruppe „DaPoXL“ wird dem bereits erwähnten gleichnamigen Excel Add-In zugeschrieben und während dessen Installation erstellt. Der Beweis der Mitgliedschaftsbedingung ist ebenfalls die der URL, welche in diesem Fall der Installationspfad ist. Da „DaPoXL“ ein Excel, also Office Add-In, ist, besitzt es volle Vertrauenswürdigkeit. Sollte ein Benutzer daraufhin die DLL des Add-Ins aus dem Installationspfad entfernen und in einen anderen Ordner verschieben, würde es nicht mehr funktionieren, da die Mitgliedschaftsbedingung nicht mehr erfüllt wird.

Damit die Einträge der Codegruppe, der Bedingung und das Setzen der Berechtigung erfolgen, muss dem Setupprojekt des Add-Ins im Visual Studio eine spezielle Installationsklasse hinzugefügt werden, die während der Installation ausgeführt wird. Dabei ist der Eintrag in die Sicherheitsrichtlinienebene des Benutzers für das Add-In ausreichend.

3.3.4 Installer-Klasse zum Setzen der Richtlinien

```
Public Overrides Sub Install(ByVal stateSaver As System.Collections.IDictionary)
    Dim user As PolicyLevel
    Dim sAssemblyPath As String = Me.Context.Parameters("DaPoXL")

    Dim policyEnumerator As System.Collections.IEnumerator
    policyEnumerator = SecurityManager.PolicyHierarchy()

    'Sicherheitsrichtlinie "Organisation"
    policyEnumerator.MoveNext()
    'Sicherheitsrichtlinie "Computer"
    policyEnumerator.MoveNext()
    'Sicherheitsrichtlinie "Benutzer"
    policyEnumerator.MoveNext()
    user = CType(policyEnumerator.Current, PolicyLevel)

    Dim fullTrust As PermissionSet = user.GetNamedPermissionSet("FullTrust")
    Dim policy As New PolicyStatement(fullTrust, PolicyStatementAttribute.Nothing)

    Dim condition As New UrlMembershipCondition(sAssemblyPath)
    Dim group As CodeGroup = New UnionCodeGroup(condition, policy)

    group.Name = "DaPoXL"
    user.RootCodeGroup.AddChild(group)
    SecurityManager.SavePolicy()

    MyBase.Install(stateSaver)
End Sub
```

Abbildung 8 - Auszug der Installer-Klasse (Install-Methode) [13]

Abbildung 8 stellt eine Methode dar, welche die ursprüngliche Installationsmethode des Setupprojektes überschreibt. Prinzipiell ändert sich an dem Installationsvorgang nichts, außer dass durch den Code dieser Methode zusätzlich Einträge in den Laufzeitsicherheitsrichtlinien erfolgen. Um den Ablauf dieser Methode während der Installation verständlicher beschreiben zu können, werden in *Tabelle 4* die wesentlichen Variablen näher erläutert.

Variablen	Beschreibung
sAssemblyPath	Ist eine Zeichenkettenvariable, die über den Parameter „DaPoXL“ definiert wird und den Installationspfad einschließlich des Dateinamens enthält.
policyEnumerator	Die Variable „policyEnumerator“ stellt ein Abbild der Struktur der Sicherheitsrichtlinienhierarchie dar.
user	Repräsentiert die Richtlinienebene des Benutzer
fullTrust	Stellt den zu verwendenden Berechtigungstyp dar.
policy	Stellt die Anweisung der Codegruppe dar.
condition	Definiert die Mitgliedschaftsbedingung bestehend aus der URL des Installationspfades
group	Stellt die Codegruppe dar

Tabelle 4 - Wesentliche Variablen der Install-Methode

Die Zeichenkettenvariable „sAssemblyPath“ wird angelegt und bekommt den Installationspfad des Add-In zugewiesen. Das durch „policyEnumerator“ angelegte Hierarchieabbild der Richtlinien startet in der Ebene der Organisation und springt mit dem Befehl „policyEnumerator.MoveNext()“ auf die nächste darunterliegende Ebene des Computers. Durch die nochmalige Ausführung des Befehls ist die Benutzerebene erreicht und die Variable „user“ wird als Zeiger auf die Benutzerrichtlinie verwendet. Um die notwendigen Anweisungen der zu erstellenden Codegruppe des Add-Ins zu definieren, wird der Berechtigungssatz der vollen Vertrauenswürdigkeit mit der Variable „fullTrust“ angelegt und der Codegruppenanweisung „policy“ zugewiesen. Mit der Mitgliedschaftsvariable „condition“ wird der Installationspfad als zu erbringende Bedingung festgelegt.
Der als „group“ angelegten neuen Codegruppe werden die nun definierten Anweisungen und Mitgliegschaftsbedingungen von „policy“ und „condition“ übergeben. Der Codegruppennamen „DaPoXL“ wird gesetzt und die Codegruppe über den Befehl „user.RootCodeGroup.AddChild(group)“ der Richtlinie des Benutzers hinzugefügt. Mit dem Befehl „SecurityManager.SavePolicy()“ werden die neu definierten Einträge den Sicherheitsrichtlinien gespeichert.

3.4 Bereitstellung erforderlicher Komponenten

Mit der Entwicklung eines Office Add-In unter Visual Studio liegt dem Add-In auch eine bestimmte Version des .NET-Framework und Visual Studio Tools for Office zugrunde. Durch diese Laufzeitumgebungen werden einem Add-In Methoden, Funktionen und Eigenschaften bereitgestellt (API), damit Steuerelemente vorhanden sein und Interaktionen ausgeführt werden können. Aus diesem Grund müssen die Voraussetzungen für das Add-In auf dem Zielcomputer gewährleistet werden.

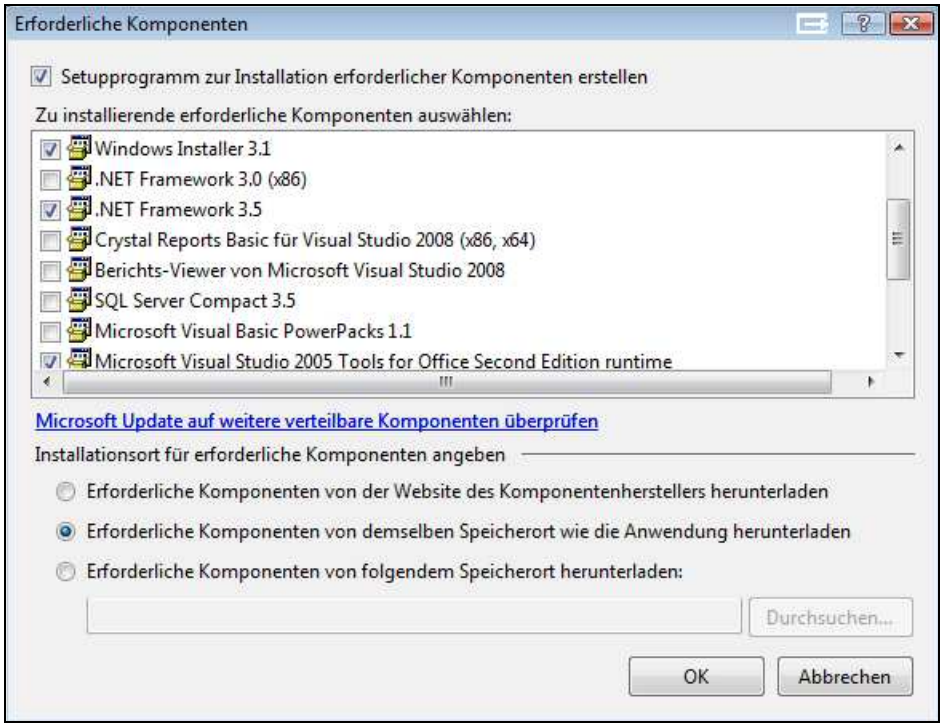


Abbildung 9 - Auswahl erforderlicher Komponenten

Dem Setup-Projekt eines Add-In können über dessen Eigenschaften erforderliche Komponenten, wie das .NET-Framework und VSTO, zugewiesen werden. In *Abbildung 9* ist das Konfigurationsfenster für die erforderlichen Komponenten dargestellt. Ein Entwickler hat die Möglichkeit diese Komponenten gepackt mit dem Setup zu liefern oder auf diese, mittels der Angabe einer Internetseite bzw. eines anderen Speicherortes (z.B. Intranet), zu verweisen. Dadurch erfolgt während der Ausführung des Setup eine Prüfung auf die notwendigen erforderlichen Komponenten. Sind diese auf dem Zielsystem nicht existent, werden sie mit installiert.

4. Das Excel 2003 Add-In „DaPoXL“

4.1 Allgemeines

4.1.1 Aufgabenstellung

Der Auftrag zur Entwicklung dieses Add-In wurde vom kaufmännischen Leiter der Schloz Wöllenstein GmbH & CO. KG erteilt. Die Basis der Idee ging von verschiedenen, selbstentwickelten Einzellösungen unter VBA aus, welche zusammengefasst werden sollten.
Unter Verwendung der Entwicklungsumgebung Visual Studio 2008 und der Programmiersprache VB.NET sollte das neu zu entwickelnde Add-In folgende Anforderungen erfüllen:

- Verwendung von mehreren Datenbanken
- Grafische Oberfläche zur einfacheren Handhabung
- Möglichkeit zur Datenbankabfrage anhand einer Schritt-für-Schritt-Zusammenstellung
- Datenbankabfrage über Excelfunktion
- Excelfunktionen sollen dynamisch und datenbankspezifisch angelegt werden können
- Administrationsbereich, um Benutzern bestimmte Datenbanken zuzuweisen
- Speichern und Laden von Abfragen, sowie XML-Dateien

Anhand dieser Anforderungen soll das zu entwickelnde Add-In benutzerfreundlich und einfach zu verwalten sein, sowie in möglichst vielen Abteilungen des Unternehmens der Schloz Wöllenstein GmbH & Co. KG zum Einsatz kommen können.

4.1.2 Begriffserklärung und Anwendungsaufbau

Der Begriff „DaPoXL“ ist ein Akronym und steht für „Datenpool für Excel“. Das Add-In dient dabei der Integration verschiedener Datenbanksysteme in Excel. Dadurch ist es möglich kompakte Auswertungen aus unterschiedlichen Datenbanken, unter Verwendung einer grafischen Oberfläche (Windows-Form) oder durch direkte Interaktion mit einer Excelzelle, anzufertigen.
Der Aufbau von DaPoXL ist daher in zwei Bereiche zu differenzieren. Es besteht aus einem COM-Add-In und aus einem Automatisierungs-Add-In, die in zwei separaten DLL-Dateien unterteilt sind. Der Teil des COM-Add-In dient der Administration und dem Anlegen von Datenbanken und Funktionen, die für die Datenbankabfragen durch eine grafische Oberfläche genutzt werden.
Das Automatisierungs-Add-In implementiert eine einzelne Excelfunktion, die der direkten Interaktion mit Excelzellen dient und ebenfalls auf die administrativ erstellten Funktionen und Datenbanken zugreift.
Die Speicherung der verschiedenen dynamischen Daten erfolgt in XML-Dateien, die von COM- und Automatisierungs-Add-In genutzt werden.

Wird im folgenden Text von Add-In oder DaPoXL gesprochen, so ist die Gesamtheit der Applikation gemeint. Anderen Falls kommt der Ausdruck des COM- oder des Automatisierungs-Add-In zur Verwendung.

4.1.3 Die Datenbanken

Datenbanken als zentrale Speichermedien werden in den meisten Unternehmen zur Verwaltung diverser Daten verwendet. Einer Datenbank liegt laut [4] ein Datenbanksystem zugrunde, in der die Datenobjekte in verschiedenen Tabellen abgelegt werden. Die Tabellen wiederum bestehen aus Spalten, deren Spalteninformation einem bestimmten Datentyp (z.B. Text oder Zahlen) entsprechen muss. Auf diese Weise werden die Daten entsprechend ihres Typs in den Spalten gespeichert. Durch die Datenbanksprache SQL (Structured Querry Language) ist es möglich Daten abzufragen, zu ersetzen, hinzuzufügen und zu löschen.

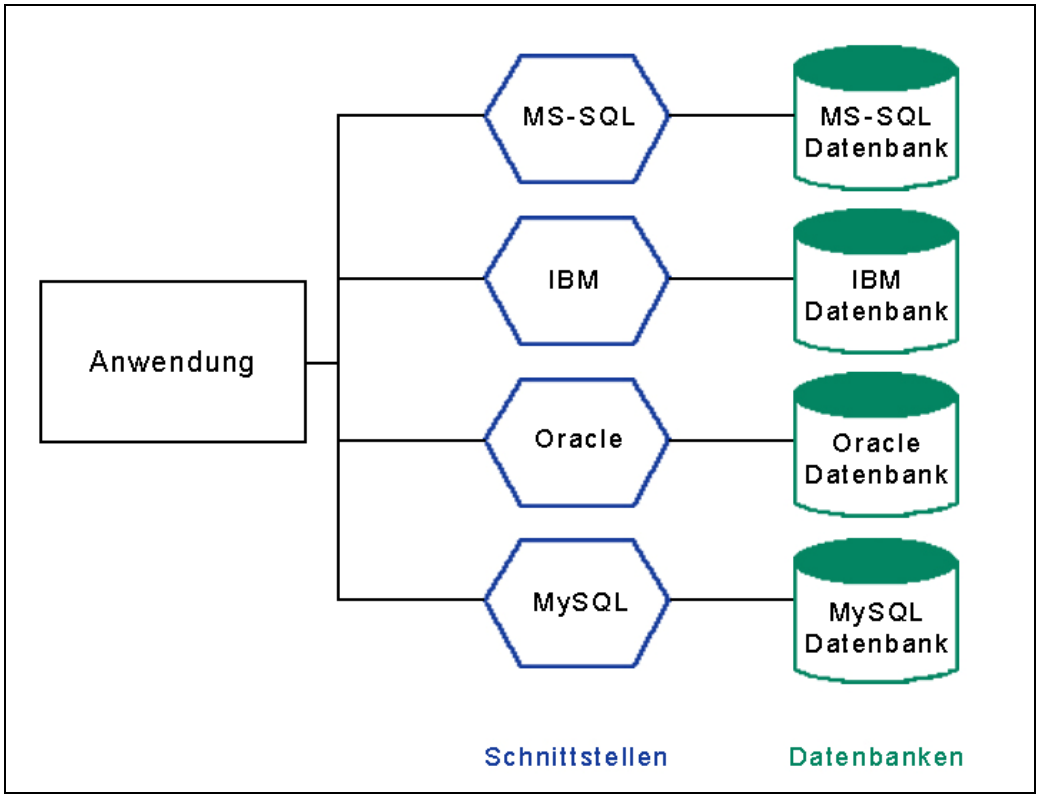


Abbildung 10 - Datenbankzugriff über Schnittstellen

Jedoch dient das Add-In DaPoXL ausschließlich der Abfrage von Daten und lässt in keiner Form die Manipulation der Daten zu. Anfangs war es angedacht DaPoXL direkt auf die verwendeten Datenbanken der Buchhaltung der Schloz Wöllenstein GmbH & Co. KG auszurichten. Dieser Ansatz wurde jedoch auf die Einbindung aller vorliegenden Datenbanken erweitert. Der Zugriff auf die Datenbanken erfolgt anhand der ihnen zugrunde liegenden Datenbanksysteme, die bei der Schloz Wöllenstein GmbH & Co. KG verwendet werden. Aus diesem Grund sind die

Schnittstellen für die Datenbanksysteme Oracle, IBM DB2, Microsoft SQL und MySQL in DaPoXL integriert worden (*siehe Abbildung 10*). Dadurch ist es dem Nutzer des Add-In möglich die verschiedenen Datenbanken dieser Datenbanksysteme abzufragen und unterschiedliche Daten in Excel übernehmen zu können.

Eine nähere Aufschlüsselung dieses Themas erfolgt unter Punkt *4.3 Die Datenbankverbindung*.

4.1.4 Die XML-Dateien

Die Extensibe Markup Language (XML) ist eine vom World Wide Web Consortium (W3C) spezifizierte Auszeichnungssprache. Sie dient nach [14] der Darstellung strukturierter Informationen auf Basis eines einfach gehaltenen Textformates. XML beschreibt lediglich die Darstellung und Struktur von Datensätzen, gibt aber nicht deren Verarbeitungsweise vor.

```
<?xml version="1.0" encoding="utf-8" ?>                    <!-- XML-Deklaration -->
<Beispiel>                                                 <!-- 1. Ebene -->
  <Beschreibung>Das ist ein Benutzerbeispiel.</Beschreibung>  <!-- 2. Ebene -->
  <!-- Benutzer mit dem Attribut "Alter" -->
  <Benutzer_1 Alter="34">
    <Vorname>Max</Vorname>                                 <!-- 3. Ebene -->
    <Nachname>Mustermann</Nachname>
  </Benutzer_1>
  <Benutzer_2 Alter="geheim">
    <Vorname>Susi</Vorname>
    <Nachname>Sorglos</Nachname>
  </Benutzer_2>
</Beispiel>
```

Abbildung 11 - Einfaches Beispiel einer XML-Datei

Abbildung 11 zeigt ein Beispiel einer XML-Datei, die in zwei Bereiche unterteilt ist. Nach [3] existiert der sogenannte Prolog, der hier als XML-Deklaration angegeben ist. Die XML-Deklaration definiert das Dokument als XML-Dokument. Darin können unter Anderem die zu verwendete XML-Version und die Zeichenkodierung angegeben werden. Darüber hinaus können dem Prolog weitere Angaben, wie zum Beispiel eine Document Type Definition (DTD) hinzugefügt werden. Die DTD dient der Angabe eines bestimmten Formataufbaus des XML-Dokuments.
Des Weiteren muss ein XML-Dokument ein Wurzelelement enthalten, welches die erste Ebene der Hierarchie ist und alle weiteren Elemente, wie es beispielsweise in *Abbildung 11* dargestellt ist, beinhaltet.

Aus Gründen der Übersichtlichkeit und Strukturierungsmöglichkeit werden zur Datenverwaltung des Add-In „DaPoXL“ XML-Dateien verwendet. Diese dienen in erster Linie zur Speicherung von Benutzerinformationen, Datenbankendaten, erstellten Funktionen und Abfragen. Das Auslesen, Erstellen und Ändern der verwendeten XML-Dateien erfolgt durch Methoden verschiedener Programm-Klassen. Diese lesen beispielsweise beim Start einer grafischen Oberfläche die jeweilige XML-Datei ein und schreiben die gelieferten Daten in die dafür vorgesehenen Steuerelemente (z.B. Textfeld).

4.2 Die Softwarestruktur

4.2.1 Grundlegendes

Für die Entwicklung des Add-In DaPoXL wurde Visual Studio 2008 und die objektorientierte Programmiersprache VB.NET verwendet, welche auf dem .NET-Framework basiert. Sie bietet unter Anderem einen leicht nachzuvollziehenden Quellcode, klare Anweisungsstrukturen und automatische Vervollständigung des Codes mit VS2008.
Eine objektorientierte Anwendung ist aus Klassen und Objekten aufgebaut. Eine Klasse ist ein Bauplan für Objekte bestehend aus Methoden und Variablen, die das Verhalten und die Eigenschaften eines Objektes kennzeichnen. Eine Klasse beschreibt, wie ein Objekt aussehen soll. Ein Objekt wiederum ist eine Instanz, also ein definiertes Abbild, einer Klasse. [1]
Um die Zusammenhänge zu verdeutlichen, bietet sich sinnbildlich die Herstellung eines Autos an.

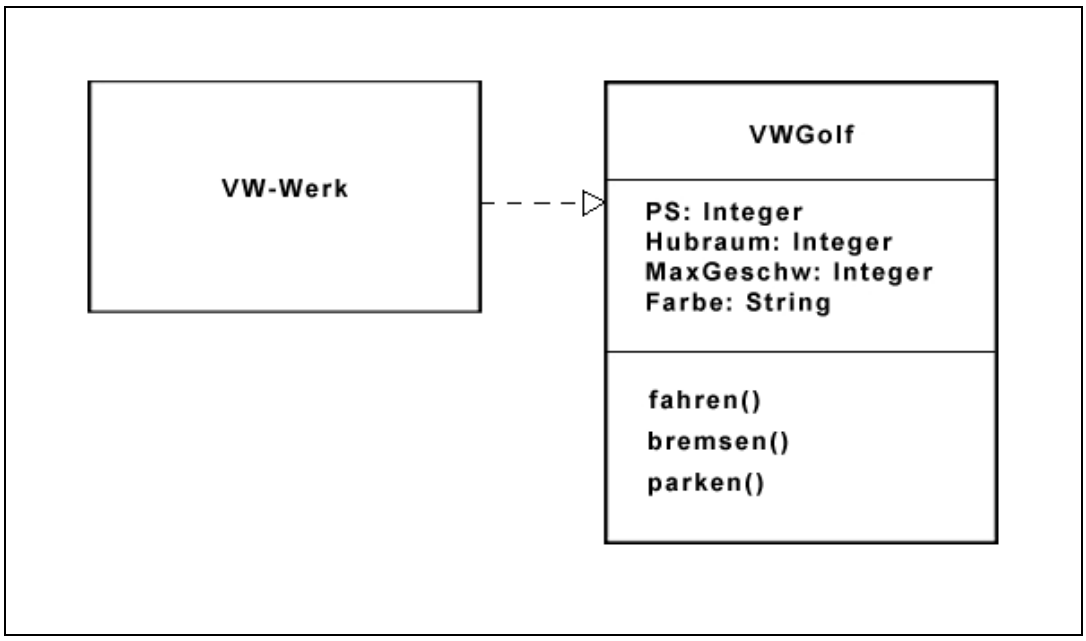


Abbildung 12 - Zusammenhang Klassen und Objekte

In *Abbildung 12* ist ein vereinfachtes Klassendiagramm mit den Klassen „VW-Werk“ und „VWGolf“ dargestellt. „VW-Werk“ ist in diesem Beispiel die Hauptklasse und produziert (instanziert) anhand eines Bauplans (Klasse „VWGolf“) einen VW Golf (Objekt). Dieser VW Golf hat in dem Beispiel diverse Eigenschaften

(Variablen), wie Anzahl an PS als Zahlenwert (Integer) und die Farbe als Zeichenkette (String). Des Weiteren besitzt der VW Golf die Verhaltensweisen (Methoden) „fahren“, „bremsen“ und „parken“, die ausgeführt werden können.
Mit einem definierten Bauplan ist es jedoch möglich, mehrere VW Golf zu produzieren, die unterschiedlich schnell sind oder eine andere Farbe haben. Anhand dieses Schemas können daher mehrere, methodisch gleiche Objekte mit unterschiedlichen Variablen instanziert werden.

4.2.2 Die Struktur des COM-Add-In

Das Projekt des COM-Add-In von DaPoXL ist aus mehreren Klassen aufgebaut. Einige dieser Klassen werden durch VS2008 beim Anlegen eines Projektes automatisch erstellt und vorkonfiguriert. So wird beispielsweise eine Klasse namens „*MySettings*“ erstellt, welche die Grundeinstellungen des jeweiligen Projektes beinhaltet. Bei Erstellung eines Excel Add-In-Projekts wird, neben der allgemeinen „*MySettings*“-Klasse, die Klasse „*ThisAddIn*“ angelegt.
Diese „*ThisAddIn*“-Klasse wird laut [15] vom Visual Studio Tools for Office generiert und dient als Kommunikationsverbindung zwischen der Microsoft Office-Anwendung und dem Programmcode des Add-In. Visual Studio Tools for Office instanziert diese Klasse automatisch, wenn die Office-Anwendung das Add-In lädt.
Über diese Klasse kann auch auf das Objektmodell der jeweiligen Office-Anwendung zugegriffen werden, um etwa Excelzellen mit Werten zu befüllen.

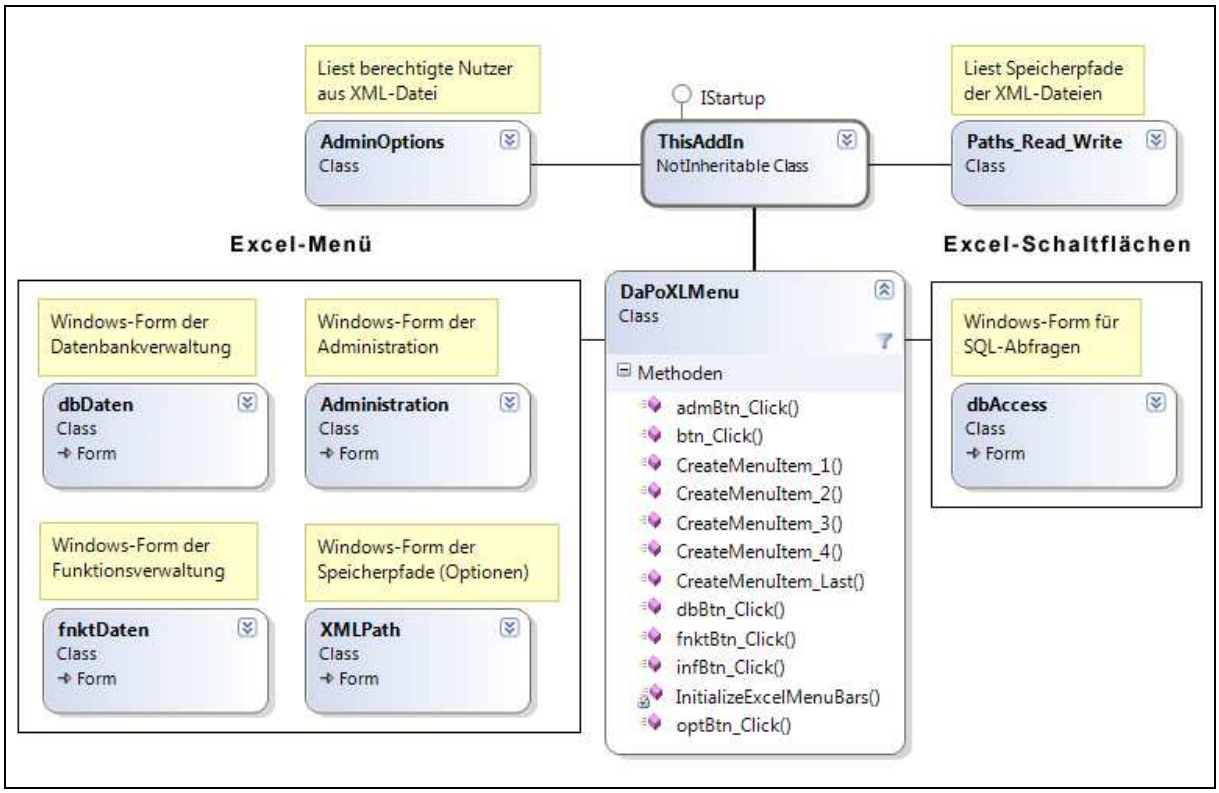


Abbildung 13 - Softwarestruktur des COM-Add-In

Abbildung 13 stellt die Klassen dar, die für die Instanzierung des in Excel integrierten Menüs, der Schaltflächen und deren Verhalten verantwortlich sind.
Die „*ThisAddIn*“-Klasse instanziert die Klassen „*AdminOptions*“, „*Path_Read_Write*“ und „*DaPoXLMenu*“. Zu Beginn wird mittels der „*Path_Read_Write*“-Klasse eine XML-Datei ausgelesen, welche die Pfade der exportierbaren XML-Dateien beinhaltet. Sollten bis dato keine Pfade gespeichert worden sein, ist die zu lesende XML-Datei leer und es werden die Pfade der XML-Dateien im Installationsverzeichnis verwendet.
Anhand der jeweils gelesenen Pfadangaben, wird durch die instanzierte „*AdminOptions*“-Klasse die XML-Datei der berechtigten Benutzer eingelesen und gleicht den angemeldeten Windows-Nutzer mit den vorhandenen Benutzern ab. Je nachdem ob der momentane Nutzer bereits existiert, ändert sich das anzulegende Menü, für das die „*DaPoXLMenu*“-Klasse verantwortlich ist.
Die „*DaPoXLMenu*“-Klasse führt dabei die Methode „*InitializeExcelMenuBars*“ aus. Diese Methode fügt dem bereits bestehenden Excel-Menü einen weiteren Eintrag namens „DaPoXL“, sowie zwei Schaltflächen (Combobox und Button) zur Auswahl der administrativ angelegten Datenbanken hinzu. Die Untermenüpunkte des Menüeintrags „DaPoXL“ werden über die „*CreateMenuItem*“-Mehtoden erstellt.
Abbildung 14 veranschaulicht das Menü als Administrator und die Schaltflächen für die Datenbankauswahl.

Abbildung 14 - Excel-Menüeintrag als Administrator

Ist der aktuelle Windows-Nutzer als Administrator eingestuft, bekommt dieser sämtliche Menüeinträge zur Verwaltung des Add-Ins bereitgestellt. Ein normaler oder nicht existenter Nutzer sieht lediglich „Optionen“ und „Info“. Weiterhin werden dem normalen Nutzer die ihm zur Verfügung stehenden Datenbanken in die angelegte Combobox eingetragen. Ein in der XML-Datei nicht existenter Nutzer hat

dennoch die Möglichkeit über „Optionen“ andere XML-Dateien anzugeben, in der er als berechtigter Nutzer hinterlegt ist.

Die Klassen „*Administration*“ (*Administration*), „*dbDaten*“ (Datenbankverwaltung), „*fnktDaten*“ (Funktionsverwaltung), „*XMLPath*“ (Optionen) und „*dbAccess“* (zur Datenbankabfrage), nach *Abbildung* 13, sind Windows-Formen (grafische Oberflächen), die den angelegten Menüeinträgen und Schaltflächen zugrunde liegen. Diese werden durch das Klick-Ereignis des jeweiligen Eintrags bzw. der Schaltfläche instanziert und aufgerufen. Jede dieser Windows-Formen wird zur Verwaltung diverser Daten verwendet, die in den XML-Dateien gespeichert sind. Mit dem Aufruf einer Form wird eine weitere Klasse instanziert, welche unter Anderem verschiedene Methoden zur Speicherung, zum Laden und zum Löschen der XML-Daten besitzt, um den Dateizugriff zu ermöglichen.
In *Tabelle 5* sind die eingesetzten Klassen und dazugehörigen XML-Dateien gelistet.

Windows-Form-Klasse	Klasse zur XML-Manipulation	Genutzte XML-Datei
Administration	AdminOptions	adm_user.dll
dbDaten	dbDaten_Option	db_daten.dll db_typen.dll sql_operatoren.dll
fnktDaten	fnktDaten_ReadWrite	fnk_funktionsliste.dll
XMLPath	Paths_Read_Write	xml_path.xml
dbAccess	sqlDLXML	sql_direktabfrage.dll

Tabelle 5 - Struktur der Menüklassen

Allerdings werden die Klassen zur XML-Manipulation nicht ausschließlich in der nach *Tabelle 5* angegebenen Windows-Form instanziert, sondern auch in anderen Klassen der grafischen Oberflächen.
So ist es beispielweise notwendig, dass die „*dbDaten_Option*“-Klasse ebenfalls in der „*Administration*“-Klasse instanziert wird, um die angelegten Datenbanken den bestehenden Benutzern zuordnen zu können.

4.2.3 Die Struktur des Automatisierungs-Add-In

Das Projekt des Automatisierungs-Add-In besteht, im Gegensatz zum COM-Add-In, prinzipiell nur aus der Klasse „*Funktionen*“. Zusätzlich werden jedoch die „*AdminOptions*“- und „*Path_Read_Write*“-Klasse verwendet. Diese gewährleisten die Prüfung der Nutzerberechtigung des angemeldeten Windows-Nutzers mit den Einträgen in den XML-Dateien.

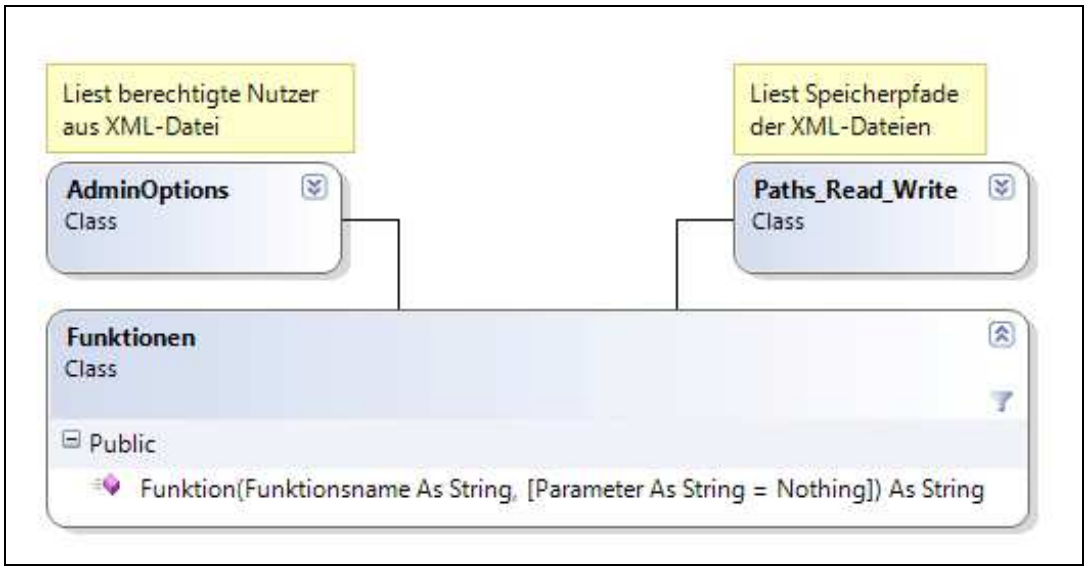


Abbildung 15 - Softwarestruktur des Automatisierungs-Add-In

Die „*Funktionen*“-Klasse (*Abbildung* 15) umfasst eine Vielzahl privater, für den Benutzer nicht verwendbarer Variablen, Methoden und Funktionen, sowie eine öffentliche Funktion. Diese öffentliche Funktion namens „*Funktion*“ wird vom Anwender genutzt, um die notwendige Datenbankabfrage zu vollziehen. „*Funktion*“ kann dabei zwei Zeichenkettenparameter entgegennehmen. Dies erfolgt über den sogenannten Function-Wizard oder über die direkte Eingabe in eine Zelle. Der erste Parameter ist der Name einer administrativ angelegten Funktion. Der Zweite entspricht den eigentlichen Parametern der erstellten Funktion und ist optional. Der Grund des zweiten optionalen Parameters dient der Bedienfreundlichkeit. Sollte der Nutzer die notwendigen Parameter der angelegten Funktion nicht kennen, sondern lediglich den Funktionsnamen, werden bei bloßer Eingabe des Funktionsnamens die zu verwendenden Parameter zurückgegeben.

Da eine reine Beschreibung der Anwendungsweise anhand der Begriffe verwirrend sein kann, soll das Beispiel auf *Abbildung 16* Klarheit verschaffen.

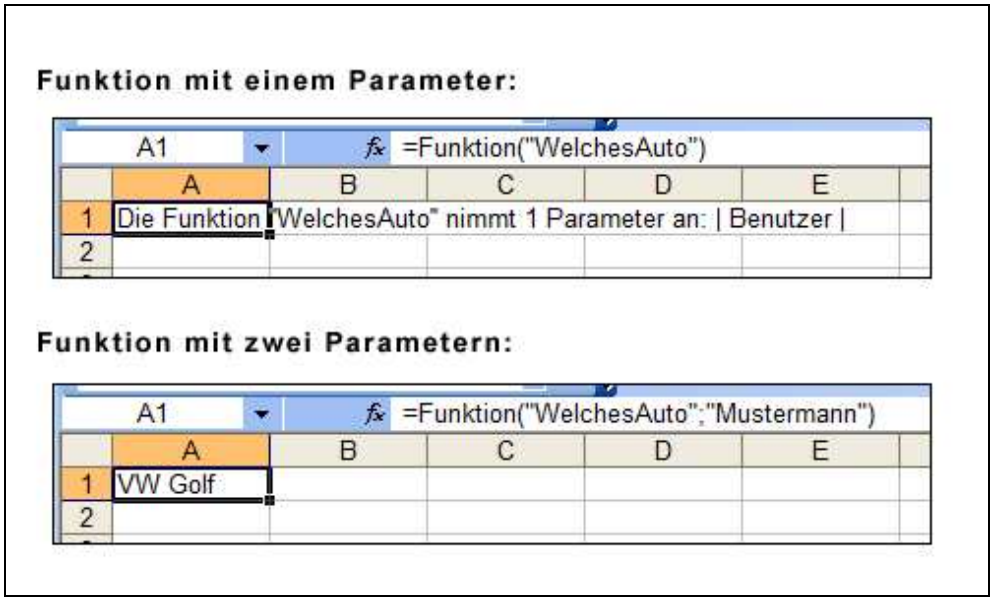


Abbildung 16 - Beispiel der Funktionsanwendung

Das abgebildete Beispiel zeigt die Verwendung von „*Funktion*“ mittels direkter Zelleneingabe. Die administrativ angelegte Funktion „*WelchesAuto“* soll die Automarke einer Person anhand ihres Benutzernamens aus der Datenbank abfragen.
Über „*=Funktion(“WelchesAuto“)*“ wird die öffentliche Funktion des Automatisierungs-Add-In angesteuert. An dieser Stelle wird innerhalb der „*Funktionen*“-Klasse die entsprechende XML-Datei mittels der „*Path_Read_Write*“-Klasse auf das Vorkommen dieser angelegten Funktion geprüft. Existiert diese, werden die zu verwendenden Parameter, hier „Benutzer“, in die ausgewählte Excelzelle geschrieben. Da der Anwender nun den Parameter der erstellten Funktion „*WelchesAuto*“ kennt, kann er den Benutzernamen hinzufügen und bekommt die Automarke aus der Datenbank zurück geliefert.

Der Grund des Prinzips der Funktion innerhalb der Funktion obliegt dabei der Registrierung des Automatisierungs-Add-In am System und der damit verbundenen Unabänderlichkeit. Durch die Registrierung werden Typ und Anzahl der Parameter einer Funktion festgelegt und sind anschließend nicht mehr veränderbar. Dadurch dient die registrierte Funktion des Automatisierungs-Add-In genaugenommen als Container, der zur Weiterverarbeitung administrativ erstellten Funktionen fungiert.

Somit ist eine Variation unter Nutzung diverser Funktionen mit beliebigen Parametern gewährleistet.

Eine detaillierte Beschreibung der „*Funktionen*“-Klasse erfolgt unter Punkt *4.5 Die Klasse der Excelfunktion und SQL-Funktionsabfrage*.

4.3 Die Datenbankverbindung

4.3.1 Die Datenbankschnittstellen

Wie bereits erwähnt, ist es mit DaPoXL möglich Datenbanken verschiedener Datenbanksysteme abzufragen. Um eine Kommunikation zwischen den Datenbanken und DaPoXL herzustellen, werden Datenbankschnittstellen verwendet. Dabei stellt in der Regel jeder Anbieter eines Datenbanksystems eine passende, maßgeschneiderte Schnittstelle für den Anwender zur Verfügung. Diese muss dann allerdings für die Abfrage auf dem jeweiligen Computer vorhanden sein. Um einer nicht vorhandenen Schnittstelle vorzubeugen, werden darum meist Datenbankschnittstellen verwendet, mit denen mehrere Datenbanksysteme abgefragt werden können.
Für Datenbankabfragen mit DaPoXL kommen drei verschiedene Schnittstellen zum Einsatz. Diese werden über verschiedene Bibliotheken des .NET-Framework eingebunden.

Datenbanksystem	Datenbankschnittstelle
MySQL	ODBC-Schnittstelle
Oracle	Oracle-Schnittstelle
Microsoft SQL Server, IBM DB2	OLE DB-Schnittstelle

Tabelle 6 - Verwendete Datenbankschnittstellen für DaPoXL

Tabelle 6 zeigt die Datenbankschnittstellen, die für die jeweiligen Datenbanksysteme verwendet werden.
Um allerdings eine MySQL-Datenbankverbindung aufbauen zu können, muss am System zusätzlich ein MySQL-Treiber installiert sein. Dieser steuert die Zugriffe unter Verwendung der ODBC-Schnittstelle. Heruntergeladen werden kann dieser von der Herstellerseite unter *www.mysql.de*. Aber um die Umständlichkeiten zu umgehen, wurde der MySQL-Treiber in der Version 5.1 dem Setupprojekt von DaPoXL hinzugefügt.
Für die OLE DB- und Oracle-Schnittstelle sind diese Vorkehrungen nicht zu treffen, da die Treiber der Datenbanksysteme bei der Schloz Wöllenstein GmbH & Co. KG vorinstalliert sind.

4.3.2 Die Art des Verbindungsaufbau

Um eine Verbindung zu einer Datenbank aufzubauen, wird neben einer Schnittstelle ein SQL-Befehl benötigt. Mit diesem Befehl wird definiert, welche Daten aus der Datenbank abgefragt werden sollen. Die Datenbanksysteme der Schloz Wöllenstein GmbH & Co. KG unterstützen alle die Datenbanksprache SQL, weisen aber eine unterschiedliche Struktur auf. Darum ist es nicht möglich einen einheitlichen SQL-Befehl für alle Datenbanksysteme zu verwenden. Stattdessen muss für jedes Datenbanksystem ein separater Aufbau des SQL-Befehls eingesetzt werden.
Um mit DaPoXL eine Datenbankabfrage vollziehen zu können, sind Schnittstellen und SQL-Befehle in vier Klassen unterteilt, die den genutzten Datenbanksystemen zugrunde liegen. Angewendet werden diese Klassen bei der SQL-Direktabfrage des COM-Add-In.

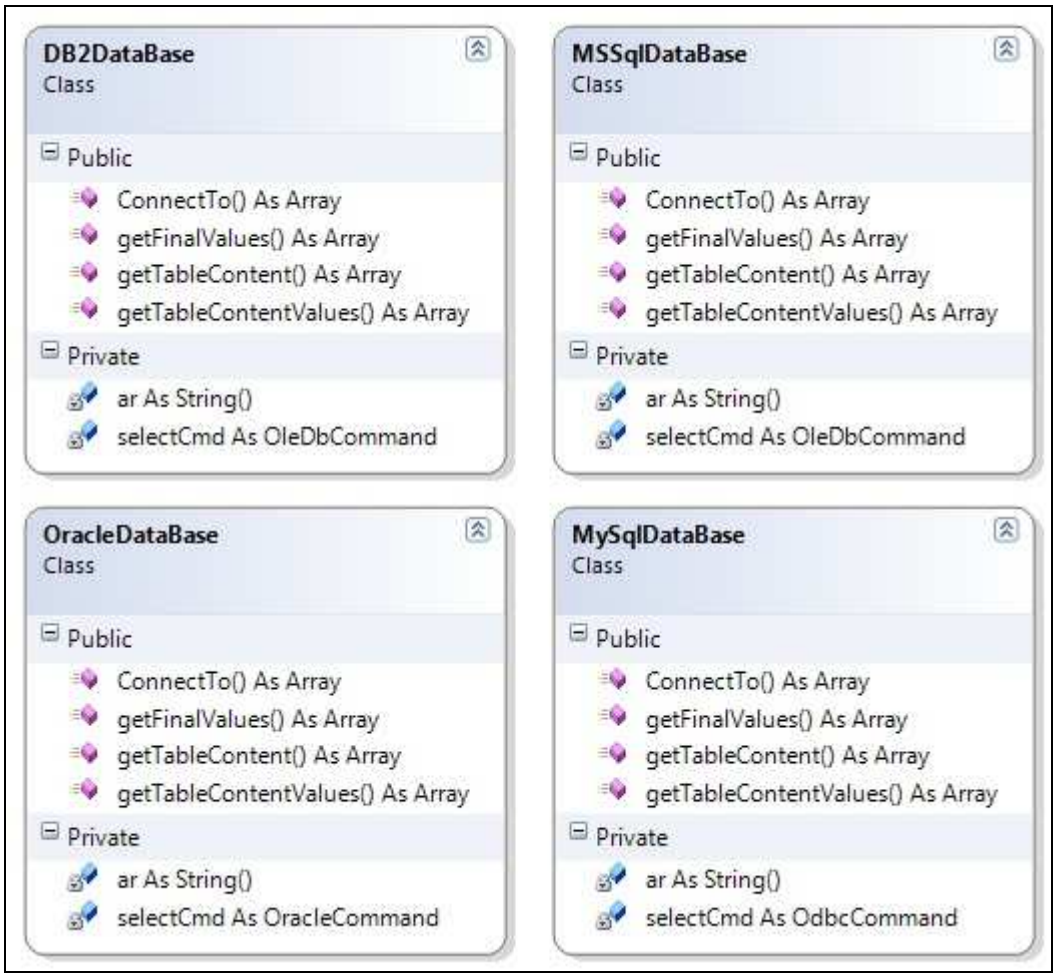


Abbildung 17 - Klassen für den Verbindungsaufbau

Die Klassen, die in *Abbildung 17* dargestellt sind, werden je nach gewählter Datenbank und dem damit bedingten Datenbanktyp instanziert. Die vier Funktionen der Klassen geben nach ihrem Aufruf eine Datenstruktur (Array), also eine Liste der abgefragten Datensätze, zurück. Der bereits angesprochene Verbindungsaufbau unter Verwendung eines SQL-Befehls erfolgt innerhalb dieser Funktionen.

Um die Unterschiede der Datenbankzugriffe zu verdeutlichen, werden im Folgenden MySQL- und Oracle-Datenbanksysteme anhand der „*getTableContent*“-Funktion verglichen. Dieser Funktion werden diverse Parameter für die Datenbankabfrage übergeben. Der zurückgegebene Datensatz spiegelt dabei die Spaltennamen einer Datenbanktabelle wieder.
Die Definition der Verbindung erfolgt durch die Datenbankschnittstelle. Je nach verwendeter Schnittstelle ist der sogenannte „*Connectionstring*“, der den Verbindungsaufbau durch verschiedene Parameter definiert, unterschiedlich.

```
'VERBINDUNGSDEFINITION UND CONNECTIONSTRING

'MySQL ODBC-Schnittstelle
Dim con As New OdbcConnection("Driver={MySQL ODBC 5.1 Driver};Server=" & dbSource & _
                              ";User=" & dbUser & ";Password=" & dbPw & _
                              ";Database=" & dbDatabase & ";Option=3;")

'Oracle-Schnittstelle
Dim con As New OracleConnection("User ID=" & dbUser & ";PWD=" & dbPw & _
                                ";Data Source=" & dbSource & ";")
```

Abbildung 18 - Datenbankschnittstelle mit Connectionstring

Durch *Abbildung 18* werden die Unterschiede der *Connectionstrings* ersichtlich. Die Übergabeparameter von Datenbankquelle (dbSource), Benutzer (dbUser) und Passwort (dbPw) werden bei beiden Schnittstellen verwendet, jedoch durch verschiedene Angabeart im Connectionstring. Die Angabe des Parameters „*dbDatabase*“ der MySQL ODBC-Schnittstelle gibt die zu nutzende Datenbank für die Abfrage vor.

Der Aufbau der SQL-Befehle, sowie der Unterschied auf Grund der Datenbanksysteme, werden hinsichtlich *Abbildung 19* deutlich.

```
'DEFINITION DER SQL-BEFEHLE

'MySQL SQL-Befehl
Dim sqlString As String
sqlString = "SELECT * FROM " & dbTable & " LIMIT 1"

'Oracle SQL-Befehl
Dim sqlString As String
sqlString = "SELECT * FROM " & dbSchema & "." & dbTable & " WHERE RowNum < 1"
```

Abbildung 19 - Angabe der SQL-Befehle

Durch die Anweisung mittels „*SELECT* *“ werden alle Spalten und Zeilen einer Tabelle abgefragt. „*FROM*“ gibt dabei die Tabelle an, die abgefragt werden soll. Die Select-Anweisung mit dem Sternchen-Operator ist bei allen für DaPoXL verwendeten Datenbanksystemen gleich, jedoch unterscheiden sich diese bei der Angabe der Tabelle.

Auf Grund der bereits vorgenommenen Zuordnung der Datenbank im Connectionstring, muss im SQL-Befehl der MySQL-Datenbank lediglich die Tabelle angegeben werden. Eine Oracle-Datenbank unterscheidet jedoch zusätzlich noch in Schemata, die der Tabelle durch einen Punkt vorangestellt werden.

Da die Funktion „*getTableContent*“ alle Spaltennamen und nicht deren Inhalte zurückgeben soll, wird der „*FROM*“-Klausel ein weiter datenbankabhängiger Befehl angehängt. Ersichtlich wird dies durch „*LIMIT 1*“ des MySQL- und „*WHERE RowNum < 1*“ des Oracle-SQL-Befehls. Beide Befehle bewirken die Einschränkung auf die erste Zeile des Datensatzes, der die Spaltennamen der Datenbank repräsentiert.

4.4 Die SQL-Direkt- und SQL-Funktionsabfrage

4.4.1 Allgemeine Erläuterung

Die SQL-Funktions- und SQL-Direkabfrage sind Begriffe, die bei der Entwicklung von DaPoXL gewählt wurden, um die Art der Datenbankabfrage kurz und passend zu umschreiben. Dabei handelt es sich bei der SQL-Funktionsabfrage um eine Möglichkeit administrativ erstellte Funktionen nutzen zu können, die einen einzelnen gezielten Rückgabewert zurückgeben (analog Funktion des Automatisierungs-Add-In). Im Gegensatz dazu dient die SQL-Direktabfrage einer Datenbankabfrage, mit der mehrere Datensätze abgefragt werden können. Der SQL-Befehl kann dabei schrittweise zusammengestellt oder über direkte Eingabe in ein Textfeld erstellt werden.

Die Klasse „*dbAccess*“ dient diesen beiden Abfragevarianten als grafische Oberfläche. Die Trennung der SQL-Funktions- und SQL-Direkabfrage erfolgt mittels „*TabControl*“-Container, der Steuerelemente aufnehmen und diese über Registerkarten verwalten kann.

Mit der Instanzierung durch die Schalflächen der Datenbankauswahl (siehe *Abbildung 14*) wird die anzuwendende Datenbank und eine Referenz der „*ThisAddIn*“-Klasse übergeben. Mithilfe dieser Referenz werden die resultierenden Datenbankabfragen in die Excelzellen geschrieben. Da durch Anwählen einer Zelle normalerweise die Windows-Form der „*dbAccess*“-Klasse in den Hintergrund von Excel verlagert werden würde, ist die Eigenschaft „*TopMost* = *True*“ gesetzt wurden. Dies bewirkt, dass sich die Oberfläche ständig im Vordergrund befindet. Dieser Vorteil, der die Handhabung zwischen DaPoXL und Excel für den Nutzer verbessert, kann allerdings auch nachteilig sein. Durch die große Anzahl an Steuerelementen, vor allem die der SQL-Direktabfrage, musste die Fenstergröße der Windows-Form auf eine Größe von 800 x 600 Pixel skaliert werden. Sollte der Anwender eine geringer Bildschirmauflösung eingestellt haben, würde die Form einen Großteil des Sichtfeldes einnehmen. Jedoch dürfte dieses Szenario normalerweise keine Rolle spielen, da hochauflösende Monitore bei der Schloz Wöllenstein GmbH & Co. KG eingesetzt werden.

4.4.2 Aufbau der SQL-Direktabfrage

Für die schrittweise Erstellung eines SQL-Befehls werden die in *Abbildung 17* dargestellten Funktionen der Datenbanksystemklassen benötigt. Diese dienen der Abfrage gewisser Werte einer Datenbank, sowie Datenbanktabelle. Der Aufruf der Funktionen erfolgt dabei einerseits mit der Instanzierung der „*dbAccess*“-Klasse und andererseits nach Interaktion mit bestimmten Schaltflächen. Die resultierenden Werte der Funktionen werden wiederum zur Befüllung verschiedener Comboboxen verwendet, um die Select-Anweisung und Where-Bedingung des finalen SQL-Befehls zu definieren. Der Strukturierung der Steuerelemente von Select- und Where-Befehl erfolgt dabei in zwei Panels (abgeschlossene Bereiche zu Gruppierung von Elementen).

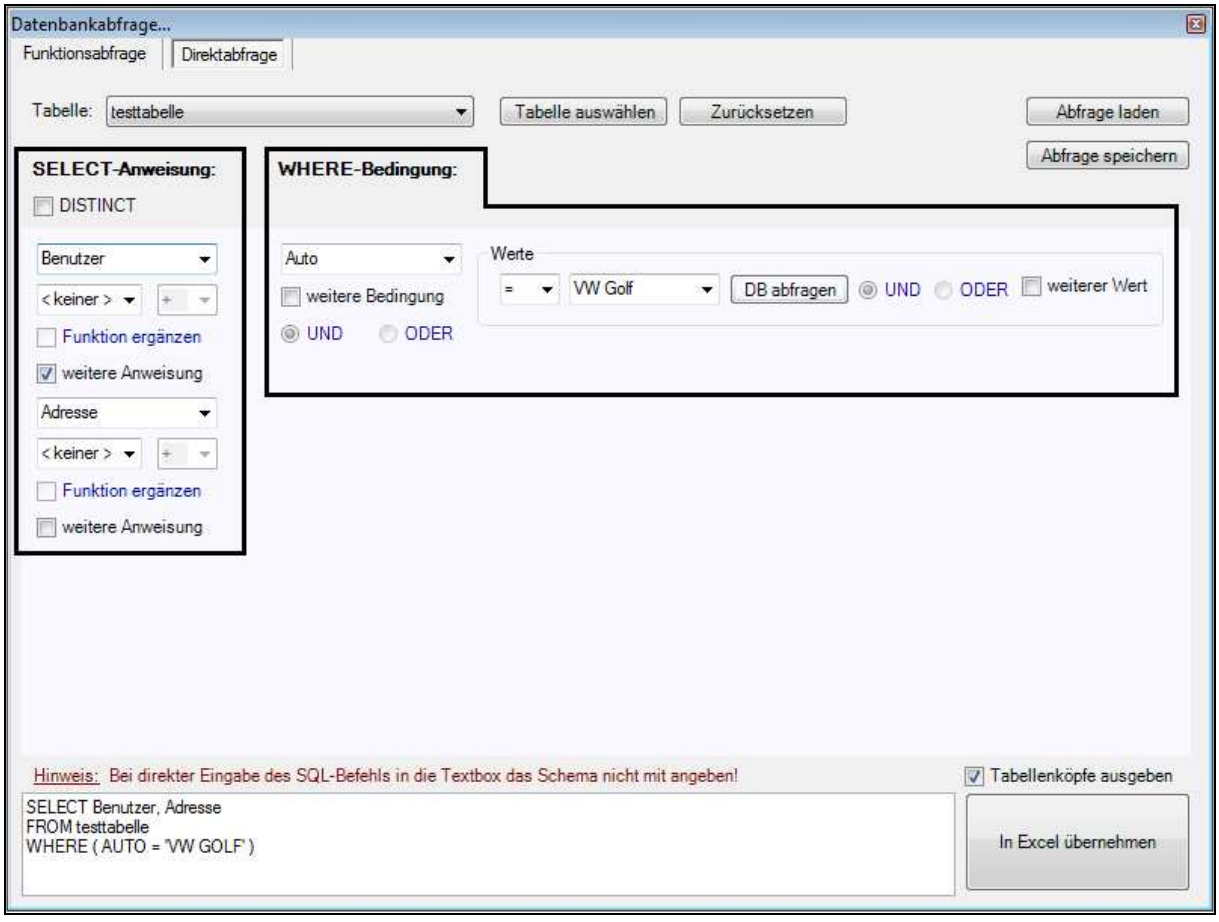


Abbildung 20 - Die Oberfläche der SQL-Direktabfrage

Abbildung 20 stellt die Oberfläche der SQL-Direktabfrage, unter Verwendung einer Testtabelle, dar. Die Steuerelemente der Select-Anweisung und Where-Bedingung werden mittels bestimmter Methoden durch das Klick-Ereignis einer Checkbox (z.B. „weitere Anweisung“-Checkbox) dem zugehörigen Panel hinzugefügt. Dabei besitzen die Elemente des gleichen Typs und Anwendungszwecks eine einheitliche Grundbezeichnung. Um sie allerdings separat voneinander ansprechen zu können, wird ihnen ein Index in Form einer Zahl angefügt. Da die Elemente einer neuen Anweisung stets die gleiche Anzahl besitzen, wird der zu verwendende Index durch Abzählen ermittelt und mit der hinzugefügten konstanten Anzahl an Elementen dividiert. Dies wird mittels folgenden Programmzeilen realisiert:

```
' Panel der SELECT-Anweisung
Dim pan As Panel = spliCon.Panel1

' Abrufen der Anzahl aller Elemente
' und dividieren durch Grundanzahl.
k = CInt(pan.Controls.Count / 5)
...
' Hinzufügen der neuen Elemente zu Panel1
' mit Index als Übergabeparameter
Panel1AddControls(k)
```

Abbildung 21 - Ermittlung der Elementindizes

Dadurch ist die Eindeutigkeit sämtlicher Steuerelemente gewährleistet. Mittels der Werte dieser definierten Steuerelemente wird der SQL-Befehl erstellt und in der auf *Abbildung 20* ersichtlichen Textbox eingefügt. Verantwortlich dafür ist eine Methode, die der Zusammenstellung des SQL-Befehls dient und mit den verschiedenen Ereignissen der Steuerelemente ausgeführt wird. Somit wird der SQL-Befehl ständig aktualisiert und passt sich den gewählten Werten der Steuerelemente an.
Wurde die „*In Excel übernehmen*“-Schaltfläche getätigt, erfolgt die finale Datenbankabfrage durch die „*getFinalValues*“-Funktion, welcher der Inhalt der Textbox als anzuwendender SQL-Befehl übergeben wird. Der Rückgabewert dieser Funktion wird in einem mehrdimensionalen Array festgehalten.

Die aktuell angewählte Excelzelle dient dabei als Startpunkt der zu übergebenden Daten. Ausgehend davon werden die Datensätze des Array durch folgenden Programmcode in Excel ausgegeben:

```
'1. Dimension des Array
For x As Integer = 0 To (dbFinalValue.GetLength(0) - 1) Step 1
    '2. Dimension des Array
    For y As Integer = 0 To (dbFinalValue.GetLength(1) - 1) Step 1
        DaPoXL.Application.ActiveCell.Cells((y + 1), (x + 1)) _
        .Value = dbFinalValue.GetValue(x, y).ToString
    Next y
Next x
```

Abbildung 22 - Auslesen der Rückgabewerte des Array

Die erste Dimension des Array variiert durch die Anzahl der gewählten Spalten (Select-Befehle). Je mehr Werte aus der Datenbank abgefragt werden sollen, desto größer ist die erste Dimension. Unter Zuhilfenahme von *Abbildung 20* hat das „*dbFinalValue*“-Array eine Länge von zwei in der ersten Dimension. Die zweite Dimension beinhaltet die zurückgegebenen Datensätze der einzelnen Select-Befehle. Deren Größe, und somit die Länge der zweiten Dimension, ist abhängig von der Anzahl der resultierenden Datensätze, welche die Abfrage zurück gibt. Anhand von *Abbildung 20* wären dies alle Datenbankeinträge, bei denen die Where-Bedingung „Auto = ’VW Golf’“ zutrifft.
Mithilfe der For-Schleifen werden somit alle Daten des mehrdimensionalen Array ausgelesen und durch die ordinate Anordnung der Excelzellen, ausgehend von der aktuell gewählten Zelle, in Excel übertragen.
Die auf *Abbildung 20* abgebildete Abfrage der Testtabelle würde folgendermaßen in Excel dargestellt werden, wenn die Zelle „A3“ ausgewählt wäre:

	A	B
1		
2		
3	Benutzer	Adresse
4	Mustermann	Feldstraße 5
5	Sorglos	Alleenweg 19
6	Niemand	Bergstraße 27
7		
8		

Abbildung 23 - Resultierender Datensatz der Beispielabfrage

4.4.3 Aufbau der SQL- Funktionsabfrage

Die SQL-Funktionsabfrage dient der Darstellung der administrativ erstellten Funktionen. Diese werden mit Instanzierung der „*dbAccsess*“-Klasse anhand der gewählten Datenbank ausgelesen und in einer Combobox zur Auswahl bereitgestellt.

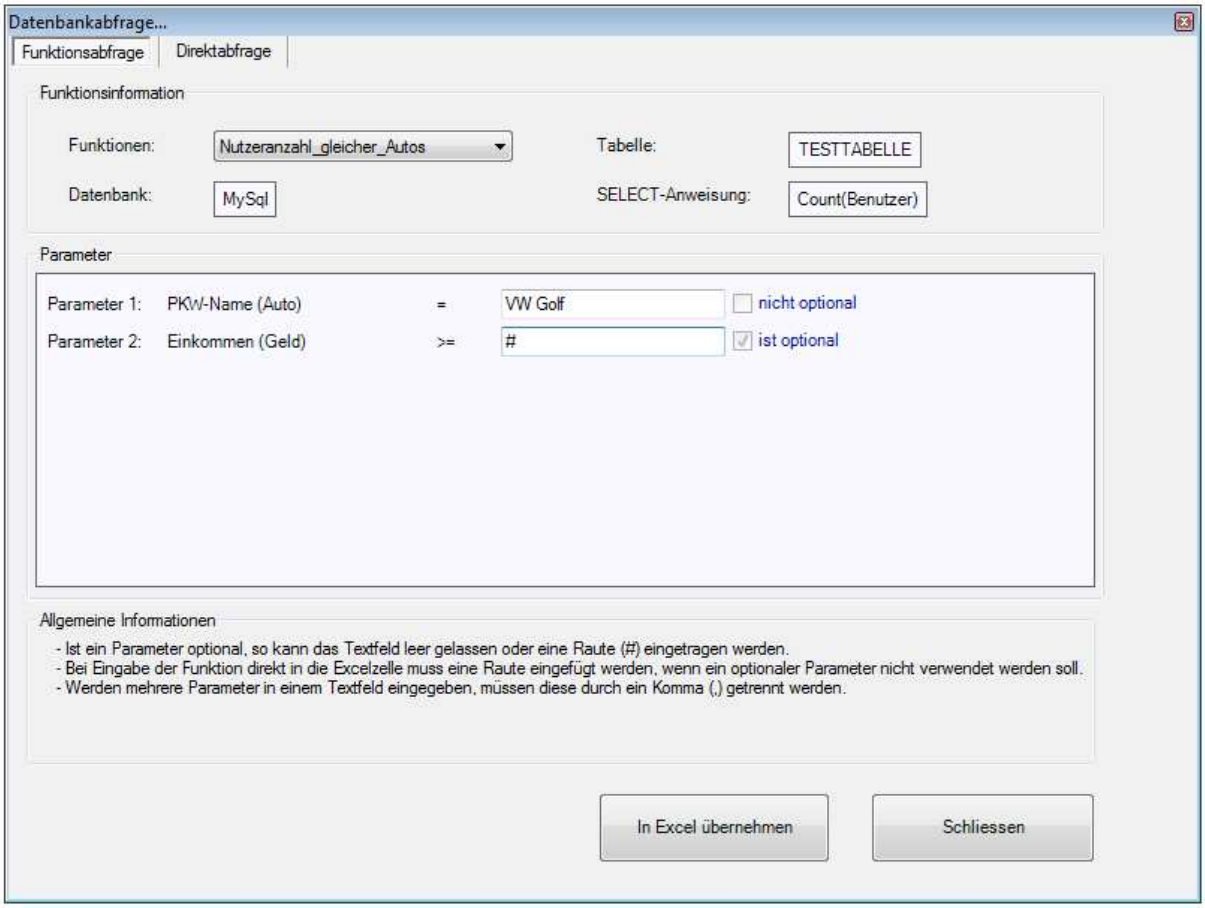


Abbildung 24 - Die Oberfläche der SQL-Funktionsabfrage

Um den Aufbau der SQL-Funktionsabfrage zu verdeutlichen, wird auf *Abbildung 24* die Oberfläche der SQL-Funktionsabfrage mittels der Beispielfunktion „*Nutzeranzahl_gleicher_Autos*“ dargestellt.

Durch die gespeicherten XML-Daten werden dem Anwender Informationen der Funktion angezeigt. Diese beinhalten die Datenbank und Tabelle die abgefragt werden, sowie die Select-Anweisung, die verwendet wird. Der Nutzer hat jedoch keine Möglichkeit die angegebenen Werte abzuändern, da diese lediglich der näheren Information dienen.

Neben den Funktionsinformationen werden die zu verwendenden Parameter in einem Panel angeordnet. Dies geschieht analog der SQL-Abfrage mit folgendem Programmcode:

```
'Bestimmung der Parameteranzahl durch Array-Länge
For i As Integer = 0 To (fkOpt.SavedParameters.Length - 1) Step 1

    'Hinzufügen der Elemente zu Panel
    'mit Index als Übergabeparameter
    PanelAddControls(i)
Next
```

Abbildung 25 - Hinzufügen der Parameter

Da die Funktionsparameter nach Auslesen der XML-Datei in einem Array zwischengespeichert sind, werden diese mit der „*PanelAddControls*“-Methode, einschließlich eines Index, dem Panel hinzugefügt. Um die Eindeutigkeit von Parameternamen sicherzustellen, kann ein Administrator bei der Funktionserstellung den Parametern Beschreibungen anfügen. In diesem Fall werden die Parameternamen den Beschreibungen in Klammern nachgestellt (*siehe Abbildung 24*). Die Werte, die ein Parameter annehmen kann, werden in der nachstehenden Textbox definiert und der späteren Abfrage übergeben.
Ist ein Parameter als optional deklariert, kann der Anwender die entsprechende Textbox leer lassen oder eine Raute (#) einfügen. Sollte die Textbox keinen Wert besitzen, wird einem optionalen Parameter die Raute über den Programmcode hinzugefügt. Angesichts der überaus seltenen Verwendung der Raute im Normalgebrauch, wurde diese zur optionalen Kennzeichnung eingebunden.

Nach Bestätigung sollte die SQL-Funktionsabfrage ursprünglich eine Zeichenkette in Form der Excelfunktion „*=Funktion(…)*“ mit den angegebenen Werten in die aktuell gewählte Excelzelle übermitteln, um daraufhin die Funktion des Automatisierungs-Add-In auszulösen. Jedoch ist diese Operation zwischen COM- und Automatisierungs-Add-In nach [7] nicht zulässig. Eine solche Interaktion verstößt gegen die Sicherheitsrichtlinien des .NET-Framework und hat die Sperrung des Automatisierungs-Add-In zur Folge.
Hinsichtlich dieses Verhaltens wurde eine identische „*Funktionen*“-Klasse des Automatisierungs-Add-In der SQL-Funktionsabfrage eingefügt. Diese Klasse wird mit

Betätigung der „*In Excel übernehmen*“-Schaltfläche instanziert und führt die Datenbankabfrage aus. Da jedoch ausschließlich der Rückgabewert ohne Hinweis auf die verwendete Funktion in die Excelzelle übertragen werden würde, wird mit folgendem Programmcode die Excelfunktion als Zellenkommentar angefügt:

```
...
' Instanzierung der "Funktionen"-Klasse
Dim fnk As New Funktionen

' Rückgabewert in aktuelle Excelzelle schreiben
' "fkAcs_cb_fkName.Text" ist Funktionsname
' "params" alle Parameter als Zeichenkette
DaPoXL.Application.ActiveCell.Value = fnk.Funktion(fkAcs_cb_fkName.Text, params)

' Kommentar der Zelle hinzufügen
' "functionString" entspricht der Excelfunktion
If DaPoXL.Application.ActiveCell.Comment Is Nothing Then
    DaPoXL.Application.ActiveCell.AddComment(functionString)
Else
    DaPoXL.Application.ActiveCell.Comment.Delete()
    DaPoXL.Application.ActiveCell.AddComment(functionString)
End If
...
```

Abbildung 26 - Programmcode der alternativen SQL-Funktionsabfrage

Daraufhin würde die auf *Abbildung 24* abgebildete SQL-Funktionsabfrage folgendermaßen in Excel dargestellt werden, wenn Zelle „B2“ aktuell angewählt wäre:

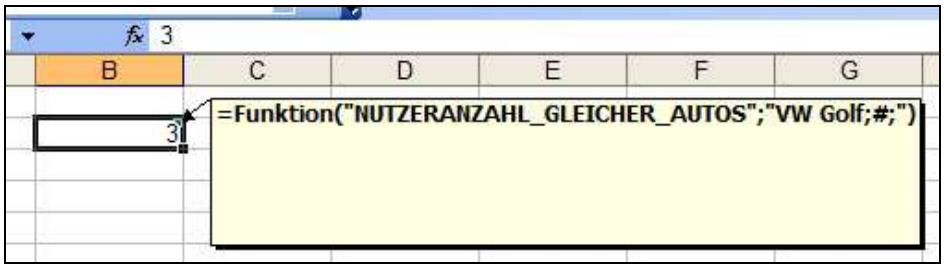


Abbildung 27 - Rückgabewert des Beispiels der SQL-Funktionsabfrage

4.5 Die Klasse der Excelfunktion und SQL-Funktionsabfrage

Die „*Funktionen*“-Klasse der SQL-Funktionsabfrage und des Automatisierungs-Add-In sind durch die verwendeten Methoden, Funktionen und Variablen, als auch der abgerufenen Daten identisch. Um den internen Ablauf einer Abfrage zu verdeutlichen, ist auf *Abbildung 28* die „*Funktionen*“-Klasse mit ihren wesentlichen Methoden und Funktionen abgebildet.

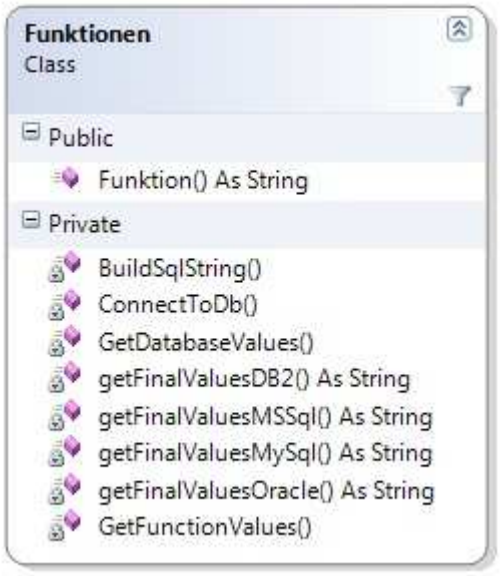


Abbildung 28 - Die „*Funktionen*“-Klasse

1. Aufruf der öffentlichen „*Funktion*“-Funktion und erste Nutzerprüfung

Eine Datenbankabfrage wird über die öffentliche Funktion namens „*Funktion*“ initiiert. Durch die bereits erwähnte „*AdminOptions*“-Klasse erfolgt eine Abfrage der gespeicherten und berechtigten Nutzer aus der XML-Datei. Sollte der aktuell angemeldete Windows-Nutzer nicht gelistet sein, wird an dieser Stelle die Datenbankabfrage abgebrochen und der nicht authentifizierte Anwender bekommt sogleich eine entsprechende Textnachricht zurückgeliefert.

2. Zweite Nutzerprüfung und Funktionsdaten abrufen

Sollte der Nutzer jedoch in der XML-Datei der Berechtigten vorhanden sein, erfolgt eine weitere vergleichbare Prüfung. Durch Ausführung der „GetFunctionValues“-Methode wird abgeglichen, ob der Nutzer berechtigt ist die zugrunde liegende Datenbank der übergebenen Funktion zu nutzen. Trifft dies zu, so werden mit dieser Methode sogleich die Werte der entsprechenden Funktion aus der

XML-Datei der Funktionsliste ausgelesen und zur Weiterverarbeitung in Variablen festgehalten. Ist dies nicht zutreffend, wird wiederum eine entsprechende Textnachricht zurückgeliefert.

3. Übergabeparameter prüfen, trennen und zuweisen

Ist der Nutzer berechtigt die Datenbank und somit die Funktion zu nutzen, erfolgt die Prüfung der zu verwendenden Parameter. Diese werden als Zeichenkette neben dem Funktionsnamen ebenfalls der öffentlichen „*Funktion“*-Funktion übergeben. Sollten jedoch keine angegeben sein, wird eine Auflistung der anzugebenden Parameter durch die Funktion zurückgegeben, wie bereits unter Punkt 4.2.3 beschrieben. Ist im Gegensatz eine Zeichenkette übergeben worden, muss diese zur Weiterverarbeitung zeichenweise ausgelesen werden, da diese die verschiedenen Parameter beinhaltet. Die Parameterzeichenkette könnte folgendermaßen aufgebaut sein:

“Wert_1a,Wert_1b;Wert2“

Die logische Trennung der Parameter innerhalb der Zeichenkette erfolgt mittels Semikolon und Komma. Dabei werden mehrere Werte eines Parameters durch Kommas und unterschiedliche Parameter durch Semikola separiert. Die in *Abbildung 29* dargestellte For-Each-Schleife ist für die Trennung und Zuweisung der Parameter in ein Array zuständig.

```
'Parameter zeichenweise auslesen
ReDim sSingleParam(fkParamVal.Length - 1)
If strng.Substring(strng.Length - 1) <> ";" Then
    strng += ";"
End If
strng = Trim(strng)
strng = strng.ToUpper
For Each c As Char In strng
    mainparam += c.ToString
    If c = ";" Then
        paramcount += 1
        If paramcount <= fkParamVal.Length Then
            mainparam = mainparam.Substring(0, mainparam.Count - 1)
            If mainparam <> "" Then
                mainparam += ","
                For Each d As Char In mainparam
                    subparam += d.ToString
                    If subparam <> "" Then
                        If d = "," Then
                            subparam = subparam.Substring(0, subparam.Count - 1)
                            ReDim Preserve sArr(i)
                            sArr(i) = subparam
                            i += 1
                            subparam = ""
                        End If
                    End If
                Next
                sSingleParam(j) = sArr
                i = 0
                j += 1
            End If
            mainparam = ""
            ...
    End If
Next
```

Abbildung 29 - Parametertrennung und Zuweisung

Die String-Variable „*strng*“ repräsentiert die gesamte übergebene Zeichenkette. Diese wird zeichenweise solange ausgelesen, bis das erste Einzelzeichen ein Semikolon ist. Die zuvor ermittelten Zeichen werden in der String-Variable „*mainparam*“ festgehalten. Anhand des Beispiels würde diese nun „Wert_1a,Wert_1b;“ beinhalten. Nach Entfernung des nachstehenden Semikolons wird das Gleiche Trennverfahren auf die „*mainparam*“-Variable angewandt. In einer zweiten For-Each-Schleife erfolgt die Trennung der in „*mainparam*“ enthaltenen Parameter anhand des Kommas. Die einzelnen Parameter werden in der String-Variable „*subparam*“ zwischengespeichert und schließlich dem Array „*sArr*“ übergeben. Sind letztlich sämtliche Parameter aus „*mainparam*“ ausgelesen, wird der Inhalt des „*sArr*“-Array einem weiteren Array zur Weiterverarbeitung übergeben.

Dieser Vorgang wiederholt sich, bis schließlich die gesamten Parameter getrennt und dem „*sSingleParam*“-Array zugewiesen sind.

4. SQL-Befehl erstellen und Datenbankdaten ermitteln

Mittels des „*sSingleParam*“-Array und der zwischengespeicherten Funktionsdaten wird der SQL-Befehl durch Aufruf der „*BuildSqlString*“-Methode erstellt. Weiterhin werden die Datenbankdaten mit der „*GetDatabaseValues*“-Methode, durch Übergabe der zu verwendenden Datenbank, ermittelt.

5. Datenbankabfrage durchführen

Durch die „*ConnectToDb*“-Methode erfolgt der Aufruf der jeweiligen „*getFinalValues*“-Datenbankfunktion anhand des zuvor ermittelten Datenbanktyps. Diese Funktionen führen mittels der Datenbankdaten und dem erstellten SQL-Befehl die Abfrage aus und geben schließlich den Rückgabewert wieder, der dem Nutzer in der Excelzelle angezeigt wird.

5. Zusammenfassung

5.1 Ergebnisse

Nach einigen Monaten Entwicklung kann der Schloz Wöllenstein GmbH & Co. KG eine erste Version des Excel 2003 Add-In übergeben werden. Durch die firmeninterne Verwendung werden nach Installation des Add-In zwei Nutzer als Administratoren gehandelt. Dies sind einerseits der kaufmännische Leiter und andererseits der Leiter der EDV-Abteilung. Diesen Administratoren obliegt es neue Nutzer oder Administratoren anzulegen, die zu verwendenden Datenbanken und Funktionen zuzuweisen, sowie die konfigurierten XML-Dateien an einen zentralen Speicherort zur Verfügung zu stellen. Eingetragene Nutzer haben dadurch die Möglichkeit, Datenbankzugriffe mittels der zwei bereits erwähnten Varianten, der SQL-Funktions- und Direktabfrage, durchzuführen.

Die Entwicklung des Add-In wurde erfolgreich abgeschlossen und alle gestellten Anforderungen erfüllt. Im Allgemeinen läuft das Add-In zur Laufzeit stabil, reagiert entsprechend der Programmierung korrekt und befindet sich damit momentan in einem umfangreichen Praxistest.

5.2 Ausblick

Auch in Zukunft wird Microsoft Excel bei der Schloz Wöllenstein GmbH & Co. KG eine Rolle spielen, aber dann sicherlich in einer neueren Version. Und sollte sich das Add-In im geschäftlichen Gebrauch durchsetzen, muss es mit den neueren Excelversionen ebenfalls funktionieren können.
Darum stellt sich die Frage der Weiterentwicklung zum Einen in die Richtung der Kompatibilität und zum Anderen des Hinzufügens weiteren Funktionalitätsumfangs.

Ein Test mit Excel 2007 hat ergeben, dass im Zusammenhang von Visual Studio eine automatische Anpassung auf die neuere Version stattfindet. Jedoch kann dadurch nicht die volle Funktionalität gewährleistet werden. Microsoft unterstützt zwar weiterhin eine Abwärtskompatibilität zu älteren Excelversionen, aber das nur in gewissem Maße. Darum muss durch stete Weiterentwicklung von Microsoft Office auch das Excel Add-In programmiertechnisch angepasst werden.

Auch der Funktionalitätsumfang des Add-In ließe sich mit Sicherheit erweitern. Beispielsweise könnte eine Oberfläche zur Kreuzabfrage zweier oder mehrerer Datenbanktabellen den Nutzen des Add-In wesentlich verbessern, da momentan mit einer instanzierten Oberfläche nur Abfragen auf eine Tabelle ausgeführt werden.

Anhang

Das Visual Studio Projekt des Excel 2003 Add-In, als auch das Setup befinden sich auf der beiliegenden CD-Rom.

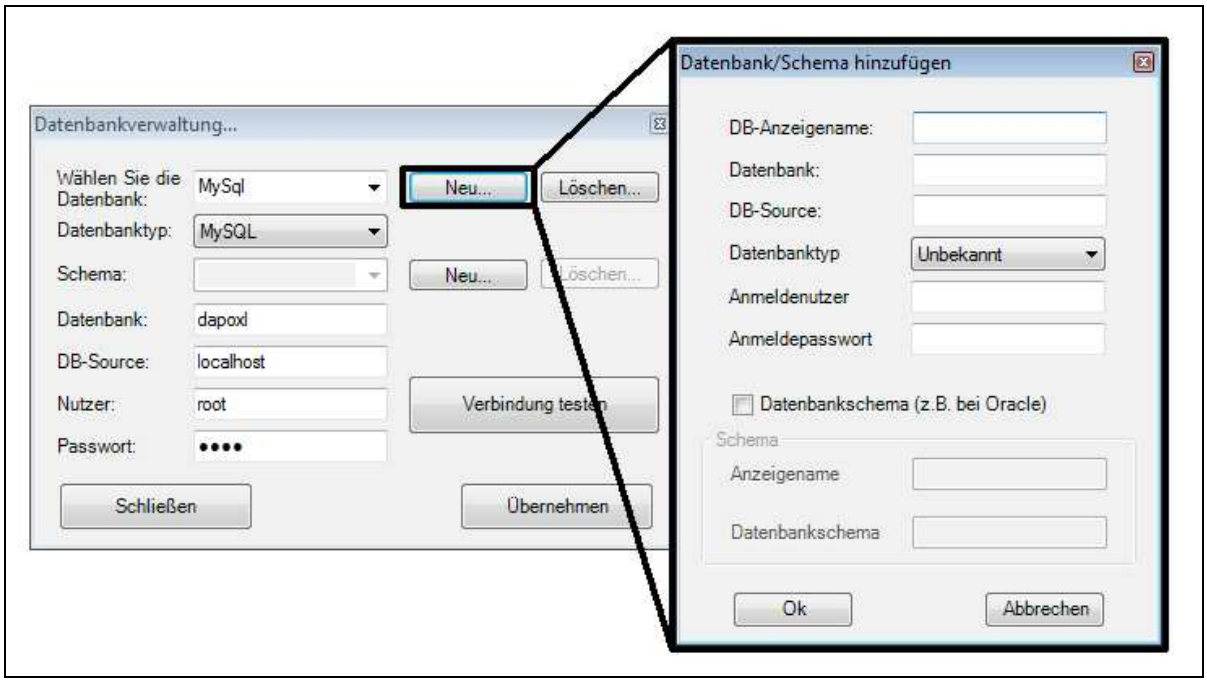
Anhang A - Kurzreferenz Datenbankverwaltungsmenü

Informationen

Zugriff von: Administratoren

Genutzt für: Erstellung, Änderung und Löschen von Datenbanken

Voreinstellungen: AlphaPlus (IBM DB2), Collectio (MS SQL), Gamis (Oracle) und FiBu (Oracle) ohne Nutzer und Passwort



Screenshot - Menüeintrag Datenbankverwaltung

Erläuterung Hinzufügung von Datenbank

1. Durch den Klick auf die „Neu“-Schaltfläche öffnen Sie das Fenster zum anlegen einer neuen Datenbank.
2. In die Textfelder geben Sie folgende Daten ein:

Textfeld	Beschreibung
DB-Anzeigename	Der Name, wie er den Nutzern angezeigt wird
Datenbank	Den Namen der Datenbank, die verwendet werden soll.
DB-Source	Die Quelle des Datenbankserver
Datenbanktyp	Den Datenbanktyp des Server
Anmeldenutzer- und Passwort	Falls benötigt Nutzername und dazugehöriges Passwort

3. Optional haben Sie die Möglichkeit ein Datenbankschema für die Datenbank anzugeben. Dies ist nicht bei allen Datenbanksystemen notwendig und kann je nach Einrichtung des Datenbankservers optional gehalten sein. Erkundigen Sie sich dahingehend bei der EDV-Abteilung oder versuchen Sie aus den voreingestellten Datenbanken die Einstellung abzuleiten.
4. Falls ein Datenbankschema erforderlich sein sollte, geben Sie folgende Daten ein:

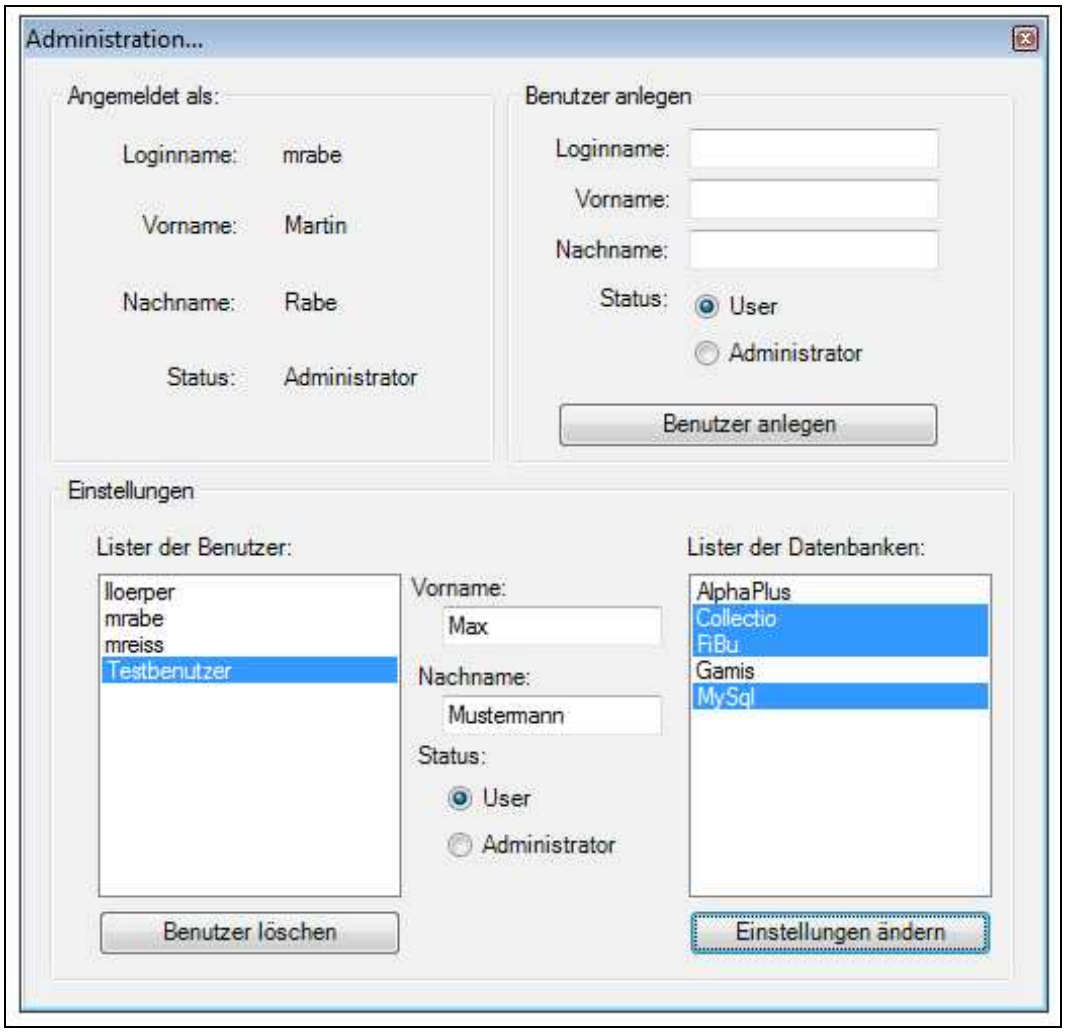
Textfeld	**Beschreibung**
Anzeigename	Der Name, wie das Schema in der Datenbankverwaltung angezeigt werden soll.
Datenbankschema	Das der Datenbank zugrunde liegende Schema.

5. Mit Klick auf die „Ok“-Schaltfläche übernehmen Sie die Werte. Sie können dann in der Datenbankverwaltung die neue Datenbank auswählen, um die Daten abzuändern, einen Verbindungstest durchzuführen oder diese wieder zu entfernen.

Anhang B - Kurzreferenz Administrationsmenü

Informationen

Zugriff von:	Administratoren
Genutzt für:	Nutzererstellung, Nutzerverwaltung, Datenbankzuweisung
Voreinstellungen:	Herr Reiß (Administrator), Herr Lörper (Administrator)



Screenshot - Menüeintrag Administration

Benutzer anlegen

1. Im oberen rechten Feld der Oberfläche unter „*Benutzer anlegen*“ haben Sie die Möglichkeit neue Nutzer anzulegen.
2. Geben Sie dazu die Daten ein und setzten Sie einen Benutzerstatus. Dabei entspricht *Loginname* dem Windowsnutzer-Namen und ist gleichzeitig ein Pflichtfeld.

3. Haben Sie die Werte eingegeben, bestätigen Sie die Eingabe mit der „*Benutzer anlegen*“-Schaltfläche. Der neue Nutzer erscheint dann in der Benutzerliste.

Benutzerverwaltung und Datenbankzuweisung

1. Um die Benutzerdaten zu ändern und die Datenbanken zuzuweisen, wählen Sie in der *Liste der Benutzer* den entsprechenden Nutzer aus.
2. Sie können Vorname, Nachname und Status durch die mittleren Steuerelemente ändern.
3. Die Datenbankzuweisung erfolgt durch Auswahl der Datenbanken in der Liste. Wählen Sie die entsprechenden Datenbanken für den Benutzer aus, die dieser verwenden darf.
4. Durch Klick auf die „*Einstellungen ändern*“-Schalftfläche übernehmen Sie die geänderten Daten für den ausgewählten Nutzer.

   → Die Änderung des eigenen Benutzerstatus von *Administrator* in *User* ist untersagt und nicht zulässig. Dadurch ist die Existenz von wenigstens einem Administrator gewährleistet.

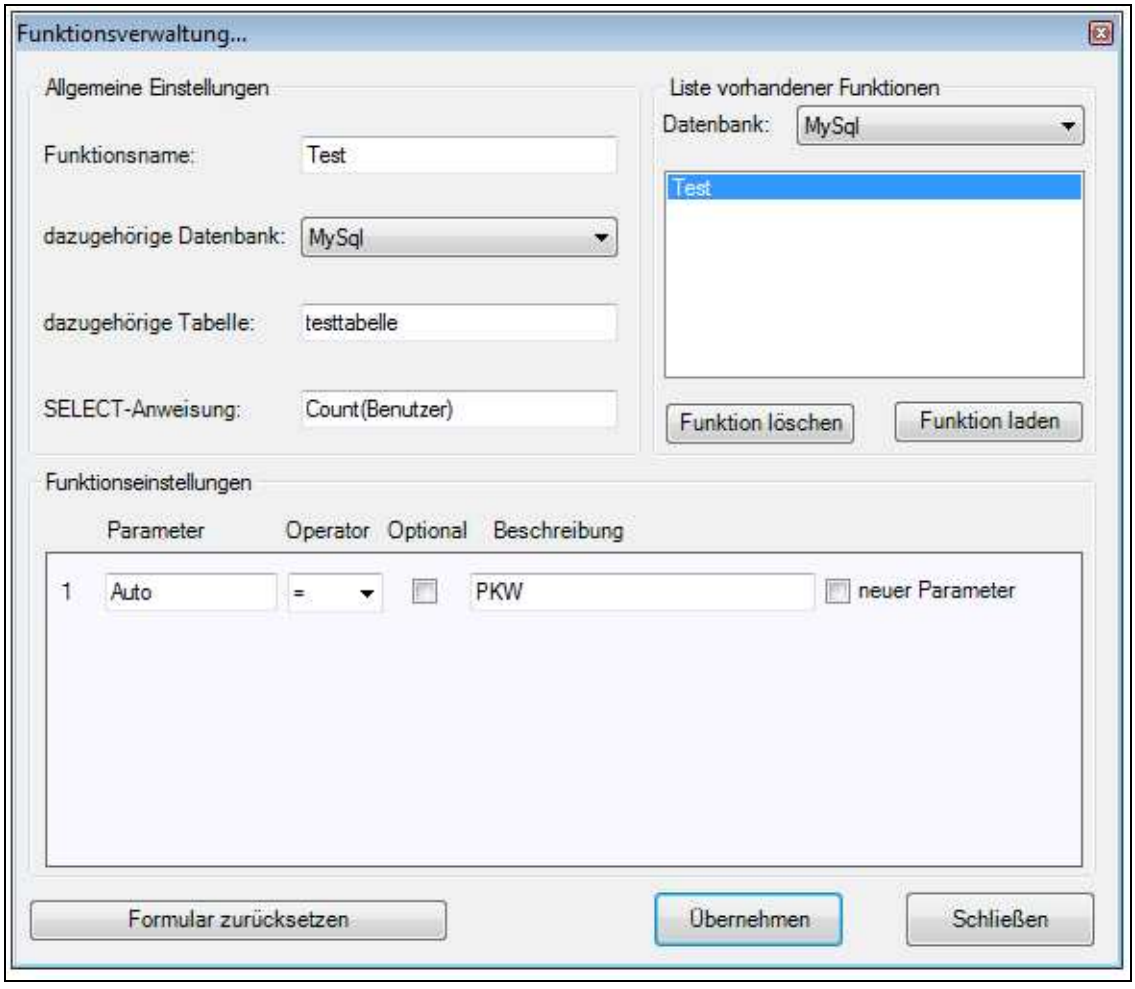
Anhang C - Kurzreferenz Funktionsverwaltungsmenü

Informationen

Zugriff von: Administratoren

Genutzt für: Funktionen erstellen, verwalten und entfernen

Voreinstellungen: keine



Screenshot - Menüeintrag Funktionsverwaltung

Funktion für Datenbank anlegen

1. Sie haben die Möglichkeit unter *Allgemeine Einstellungen* eine neue Funktion anzulegen.
2. Dabei sind folgende Einstellungen wichtig:

Eingabefeld	**Beschreibung**
Funktionsname	Der Name der Funktion, so wie er bei einer Abfrage angegeben werden soll.
Dazugehörige Datenbank	Die zugrunde liegende Datenbank
Dazugehörige Tabelle	Die zugrunde liegende Datenbanktabelle
SELECT-Anweisung	Die Anweisung, die von der Datenbanktabelle abgefragt werden soll.

3. Achten Sie darauf, dass die SELECT-Anweisung nicht mehrere Werte wiedergeben darf. Sollte dies der Fall sein, wird stets der letzte Wert übergeben werden.
4. Geben Sie unter *Funktionseinstellungen* die Parameter der Funktion an. Durch klicken auf die „*neuer Parameter*“-Checkbox fügen Sie einen Parameter hinzu oder entfernen ihn. Die Angaben zu einem Parameter sind folgendermaßen aufgebaut:

Eingabefeld	**Beschreibung**
Parameter	Der Wert der Datenbanktabelle, der abgefragt werden soll. In der Regel entspricht dieser den Spaltennamen.
Operator	Der gewünschte Vergleichsoperator, der angewendet werden soll.
Optional	Kennzeichnet den Parameter als Optional, falls der Haken gesetzt ist.
Beschreibung	Die Beschreibung soll dem Anwender eine Hilfe sein, falls der Parametername nicht eindeutig oder unverständlich ist.

5. Durch Betätigung der „*Übernehmen*“-Schaltfläche wird die neue Funktion angelegt.

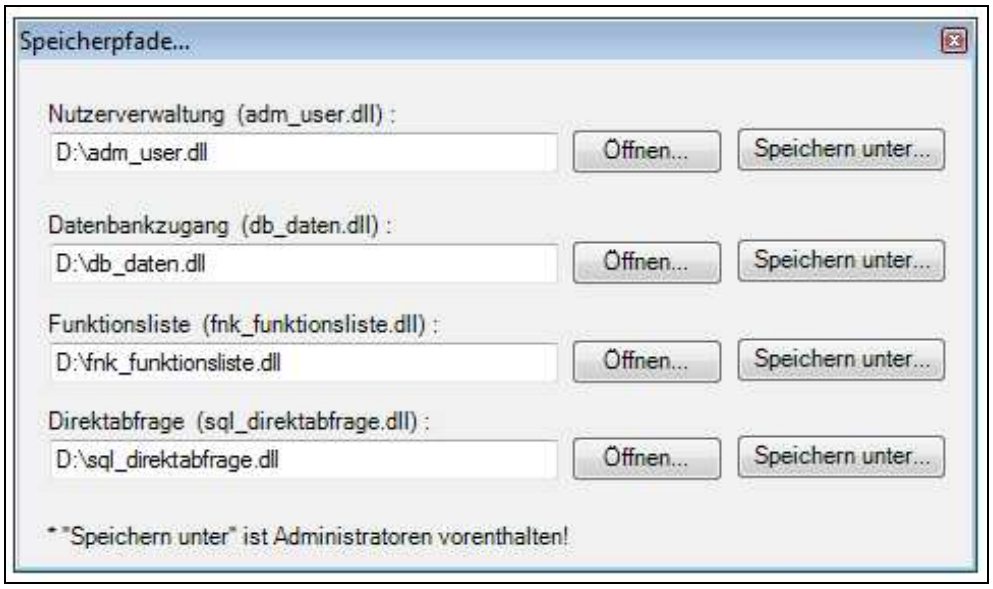
Funktion laden / löschen

1. Mittels der *Liste vorhandener Funktionen* haben Sie die Möglichkeit, die angelegten Funktionen der jeweiligen Datenbanken durchzusehen.
2. Sie können die gewählte Funktion mit der „*Funktion löschen*“-Schaltfläche entfernen oder durch Doppelklick bzw. den „*Funktion laden*“-Schaltfläche die Funktion laden, um die Daten der Funktion abzuändern.
3. Ändern Sie dabei den Funktionsnamen, wird nach der Bestätigung prinzipiell eine neue Funktion angelegt. Ansonsten werden die Werte der Funktion entsprechend abgeändert.

Anhang D - Kurzreferenz Optionsmenü

Informationen

Zugriff von:	Administratoren, normale Nutzer
Genutzt für:	Pfade der Speicherdateien
Voreinstellungen:	Installationspfad



Screenshot - Menüeintrag Optionen

Änderung der angewendeten Dateien

1. Sie haben die Möglichkeit den Pfad der jeweiligen Datei direkt in das Textfeld einzugeben oder Sie nutzen die „*Öffnen*“-Schaltflächen, um die Dateien auf Ihren Computer zu suchen.
2. Nutzen Sie die Schaltflächen, so wählen Sie die anderen Dateien aus und bestätigen Sie die Auswahl. Der neue Pfad der Datei wird in der jeweiligen Textbox angegeben.
3. Sollte sich Ihr Benutzerstatus zu *Administrator* geändert werden, werden Sie aufgefordert, Excel einmal neuzustarten. Danach stehen Ihnen sämtliche Menüeintrage zur Verfügung.

Geänderte Dateien an einen Anderen Ort speichern (nur Administratoren)

Administratoren haben die Möglichkeit, die geänderten XML-Dateien umzuspeichern. Dabei bleiben die zuvor verwendeten Dateien als Sicherung bestehen.

Literaturverzeichnis

[1] Fuchs, Joachim; Barchfeld, Andreas: Das Visual Basic .NET Codebook, München: Addison-Wesley, 2003

[2] Monadjemi, Peter; Pfeifer, Eckehard: Microsoft Office 2003. Das Entwicklerbuch: Office-Programmierung mit Net, VSTO und XML, 1. Auflage Remscheid: Microsoft Press, 2005

[3] Zimmer, Frank: Lehrveranstaltungsunterlagen Webprogrammierung II Teil XML, Hochschule Mittweida (FH)
Stand: Februar 2010

[4] Thomanek, Rico: CTI-Arbeitszeiterfassung – Spezifizierung und Implementierung. - 2001. - 127 S., Mittweida, Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik, Diplomarbeit, 2001

[5] Microsoft Corporation: Microsoft .NET Framework. URL: http://msdn.microsoft.com/de-de/library/bb979310.aspx,
Aktualisiert: 14. Juni 2004

[6] Microsoft Corporation: Übersicht über die Visual Studio Tools for Office-Laufzeit. URL: http://msdn.microsoft.com/de-de/library/bb608603.aspx,
Aktualisiert: November 2007

[7] Microsoft Corporation: Excel COM add-ins and Automation add-ins. URL: http://support.microsoft.com/kb/291392/en,
Stand: 10. Januar 2007

[8] Microsoft Corporation: Architektur von Add-Ins auf Anwendungsebene. URL: http://msdn.microsoft.com/de-de/library/bb386298.aspx,
Aktualisiert: November 2007

[9] Microsoft Corporation: Primäre Interopassemblys in Office. URL: http://msdn.microsoft.com/de-de/library/15s06t57.aspx, Aktualisiert: November 2007

[10] PCDPan_Fee: Total Registry - Infoguide rund um die Registry. URL: http://www.wintotal.de/artikel/artikel-2004/7125.html, Stand: 21.06.2004

[11] Microsoft Corporation: CLSID Key. URL: http://msdn.microsoft.com/en-us/library/aa908849.aspx, Aktualisiert: 28. August 2008

[12] Microsoft Corporation: Codezugriffssicherheit. URL: http://msdn.microsoft.com/de-de/library/930b76w0.aspx, Aktualisiert: November 2007

[13] Hansen, Steve: How Do I: Deploy a VSTO Add-In. URL: http://msdn.microsoft.com/en-us/office/bb851702.aspx, Stand: 25. September 2007

[14] World Wide Web Consortium (W3C): XML Technology. URL: http://www.w3.org/standards/xml, Gefunden: 18. Februar 2010

[15] Microsoft Corporation: AddIn-Hostelement. URL: http://msdn.microsoft.com/de-de/library/aa942742.aspx, Aktualisiert: Juli 2008

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Mittweida, den 26. März 2010 …….……