Nutzung von .Net Framework-Funktionen in einer SQLWindows-Anwendung

MD Consulting & Informationsdienste GmbH

www.md-consulting.de

Michaelisstraße 13 a
99084 Erfurt
03 61 / 5 65 93-0

Berghamer Straße 14
85435 Erding
0 81 22 / 97 40-0

info@md-consulting.de

MD Consulting & Informationsdienste GmbH

Nutzung von .Net Framework-Funktionen in einer SQLWindows[1]-Anwendung:

In SQLWindows lassen sich mithilfe des .Net Explorers nicht visuelle[2] .Net-Komponenten in die SQLWindows-Anwendungen integrieren. Beim Versuch, diese Technologie(n) für den SQLWindows-Anwender nutzbar zu machen, stellen sich möglicherweise eine Reihe von Fragen:

- Kann ich .Net-Komponenten nur in .Net WPF-Anwendungen oder auch in „klassischen“ Win32-Anwendungen nutzen?
- Wenn ich anfange, .Net-Komponenten in einer Win32-Anwendung zu nutzen, wird das eine möglicherweise später vornehmende Kompilierung meiner Anwendung als .Net WPF-Anwendung einschränken oder behindern?
- Wie finde ich mich in dem Dschungel der Funktionalitäten im .Net-Framework zurecht? Wie finde ich heraus, welche Funktionalitäten genutzt werden können?
- Wie werden die vielen Unterschiede, die es in der .Net-Entwicklung im Vergleich zu SQLWindows-Entwicklung gibt, abgebildet? (Methoden, Eigenschaften, Konstrukturen, „Überladen“, Enumerations, usw.)

Mit diesem Papier soll versucht werden, anhand einiger Beispielanwendungen und deren Dokumentation die eingangs gestellten Fragen zu beantworten.

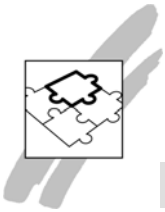
---

[1] In diesem Papier wird davon ausgegangen, dass das Vollprodukt Team Developer installiert ist. Sollte lediglich die „klassische“ Variante (ohne .Net Compiler) installiert sein, können nur die Win32-Beispiele nachvollzogen werden. Sollte in diesem Fall der .Net Explorer aus der Entwicklungsumgebung heraus gestartet werden können, dann muss er aus der Programmgruppe aufgerufen werden. Er kann dort unter der Bezeichnung .Net Explorer als eigenständiges Programm aufgerufen werden.

[2] Nicht visuelle Komponenten sind .Net Klassen, die keine eigene Oberfläche zur Verfügung stellen.

MD Consulting & Informationsdienste GmbH

Inhaltsverzeichnis



MD Consulting & Informationsdienste GmbH

Einleitung

Mit dem Vollprodukt Team Developer können „klassische“ Win32-Anwendungen, aber auch (Microsoft .Net) WPF Anwendungen („windows presentation foundation“) entwickelt werden. Obwohl sich die beiden Kompilate wesentlich u.a. in Bezug auf die Bereitstellung der Anwendung unterscheiden – eine „klassische“ Win32-Anwendung benötigt die eigene Ablaufumgebung, während eine WPF-Anwendung auf die auf jedem Windows-Rechner vorhandene .Net-Runtime aufsetzt – braucht der Entwickler bei der Erstellung seiner Anwendung diese Aspekte nicht zu berücksichtigen.

Interessant wird es allerdings, wenn der Entwickler Funktionen des .Net-Frameworks bei der Anwendungsentwicklung integrieren will. SQLWindows bietet auch diese Möglichkeit. Im Folgenden soll daher gezeigt werden, wie bei der Integration von .Net-Framework-Funktionen vorgegangen werden muss, wobei zunächst unterschieden wird, ob die Anwendung selbst als eine Win32-Anwendung oder als eine .Net WPF-Anwendung dem Anwender zur Verfügung gestellt werden soll.

MD Consulting & Informationsdienste GmbH

Aufgabenstellung 1: Speichern einer (Export-)Datei im Verzeichnis „Meine Dokumente“

In einer SQLWindows-Anwendung soll es möglich sein, Daten in Dateien zu schreiben. Damit der Anwender nach erfolgreichem Export die Dateien leicht finden kann, sollen die Dateien in den logischen Ordner „Meine Dokumente“ gespeichert werden. Da die Anwendung unter den unterschiedlichsten Windows-Betriebssystemen von den unterschiedlichsten Anwendern genutzt werden können soll, geht es also darum, den physischen Pfad des logischen Verzeichnisses „Meine Dokumente“ zu ermitteln. Eine derartige Funktion stellt das .Net Framework mit der Eigenschaft *SpecialDirectories.MyDocument* zur Verfügung. Im Folgenden soll gezeigt werden, wie diese Eigenschaft in SQLWindows-Anwendungen „eingebaut“ werden kann.

Lösungsweg

Die .Net-Dokumentation der Eigenschaft SpecialDirectories.MyDocuments kann sehr einfach über eine entsprechende Internet-Recherche (in der MSDN-Dokumentation) gefunden werden.

Wichtig zum Verständnis der Dokumentation aus Sicht des SQLWindows-Programmierers ist, dass sich die genannte Funktion SpecialDirectories.MyDocuments im Namensraum (namespace) von Microsoft.VisualBasic.FileIO befindet. Innerhalb dieses Namensraums befindet sich die SpecialDirectories Klasse.

Aus der Syntax-Dokumentation (in C#) kann abgelesen werden, dass die Funktion zum Ermitteln („get“) eines Wertes verwendet werden kann.

Wenn daher diese Eigenschaft in einer SQLWindows-Anwendung durch Aufruf einer .Net Funktion ermittelt werden soll, dann muss in der SQLWindows-Umgebung eine Schnittstelle geschaffen werden, mit der zum einen diese „entfernte“ Funktion aufgerufen und zum anderen der Datenaustausch zwischen .Net Assembly und SQLWindows-Anwendung organisiert werden kann.

Die Vorgehensweise, wie genau das erreicht werden kann, soll im Folgenden anhand des kleinen Beispiels erläutert werden.

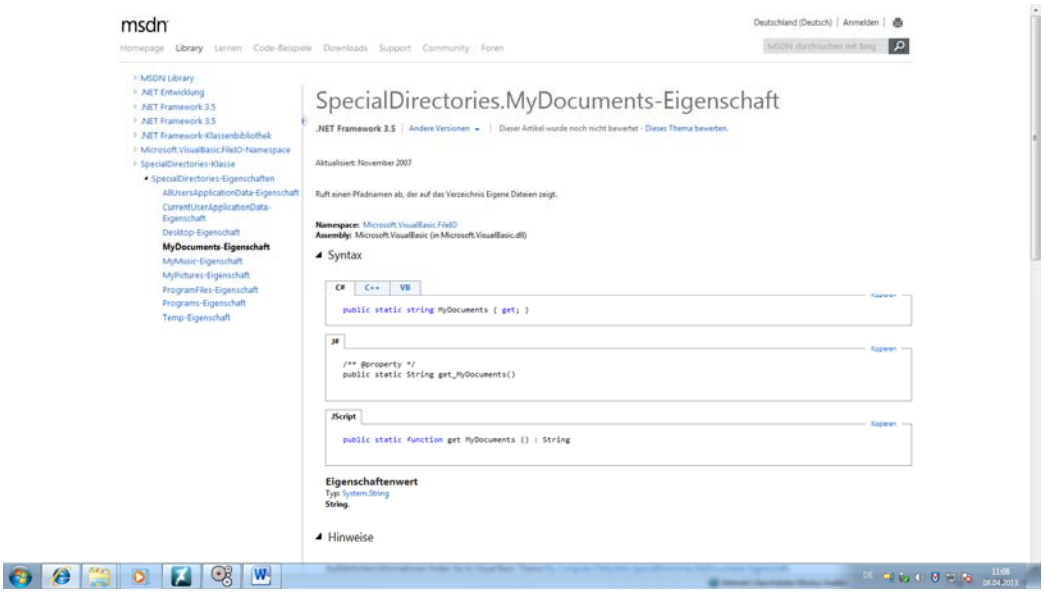


Abbildung 1: Die (Online) Dokumentation der Eigenschaft MyDocuments

MD Consulting & Informationsdienste GmbH

Beispielanwendung Win32

In der Beispielanwendung wird ein Tree View Control verwendet, um die „eigenen“ Verzeichnisse im logischen Verzeichnis „Meine Dokumente“ anzuzeigen.

Abbildung 2: Anzeige der „eigenen“ Verzeichnisse in der Anwendung

In diesem Beispiel werden sechs Unterverzeichnisse verwaltet.

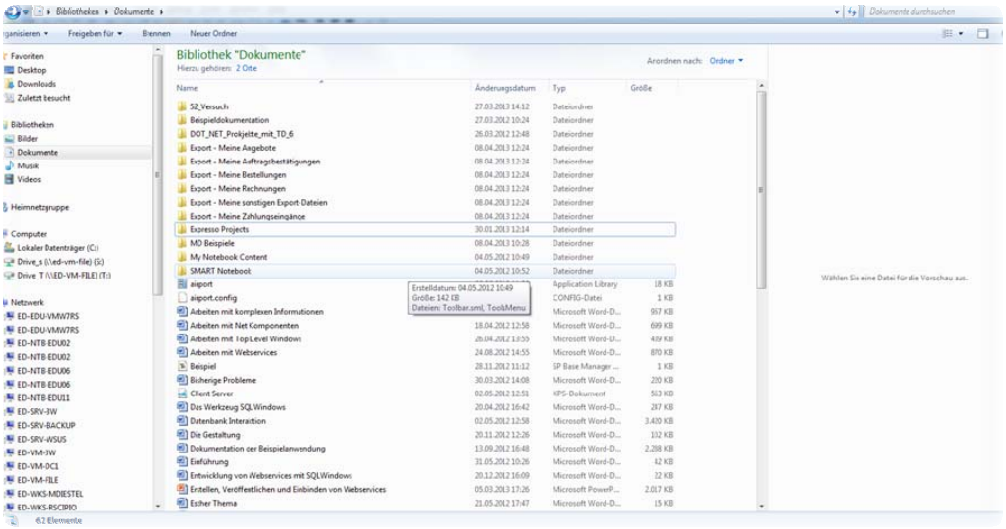


Abbildung 3: Die „eigenen“ Verzeichnisse im Dateisystem

Die von der Anwendung angelegten und verwalteten Verzeichnisse beginnen mit der Bezeichnung „Export“

Erstellung der Proxy-Bibliothek für die Integration von .Net (Framework)-Funktionen

Zunächst wird eine neue Win32 Quellcode-Datei angelegt und in einem bestimmten Verzeichnis – z.B. Beispiel – abgespeichert, da alle Bestandteile der Anwendung (Hauptdatei, Bibliotheken, Ressourcen) in einem Verzeichnis abgelegt werden sollen.

MD Consulting & Informationsdienste GmbH

Als nächstes wird der .Net Explorer (Tools, .Net Explorer) aufgerufen. Es wird der Eingangsbildschirm des .Net Explorers (in der Win32 Variante) angezeigt.

Abbildung 4: Der Eingangsbildschirm des .Net Explorers

Durch Anklicken der Schaltfläche Next gelangt man in die nächste Maske, in der der Typ der zu integrierenden .Net Assembly ausgewählt werden kann.

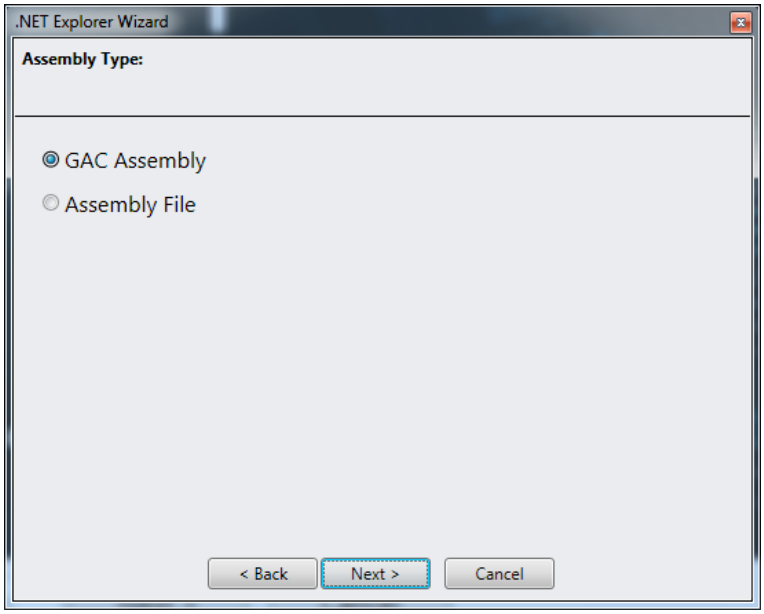


Abbildung 5: Auswahl des .Net Assembly Typs

Es können .Net Assemblies aus dem „global assembly cache“ (GAC) oder aus eine Assembly-Datei (.Net DLL) ausgewählt werden.

Da es sich im vorliegenden Beispiel um eine Funktion aus dem .Net Framework handelt und alle zum .Net Framework gehörenden Bestandteile im „global assembly cache“ verwaltet werden, wird die Option GAC ausgewählt und durch Anklicken der Schaltfläche Next in die Import-Maske geschaltet.

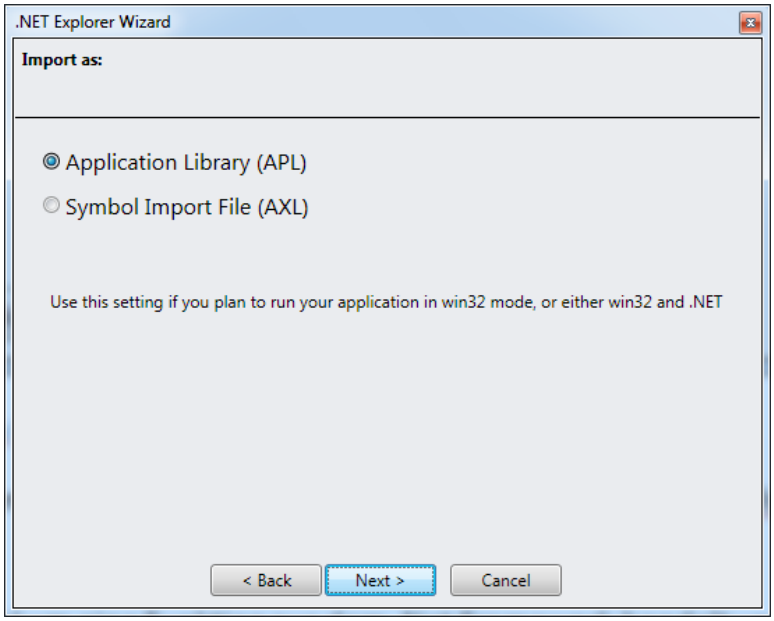


Abbildung 6: Festlegung der Importdatei

Zwei Möglichkeiten stehen zur Verfügung: es kann eine APL-Datei erstellt werden. Diese Option muss gewählt werden, wenn eine Win32-Anwendung erstellt werden soll. (In diesem Fall wird die Schnittstelle zu den .Net Funktionen über das generic application interface layer (GAIL) angebildet. Die Variante AXL wird gewählt, wenn die zu integrierende Funktionalität in eine .Net WPF Anwendung eingebettet werden soll.

In diesem Beispiel wird die Variante APL gewählt und durch Anklicken der Schaltfläche Next in den nächsten Bildschirm geschaltet.

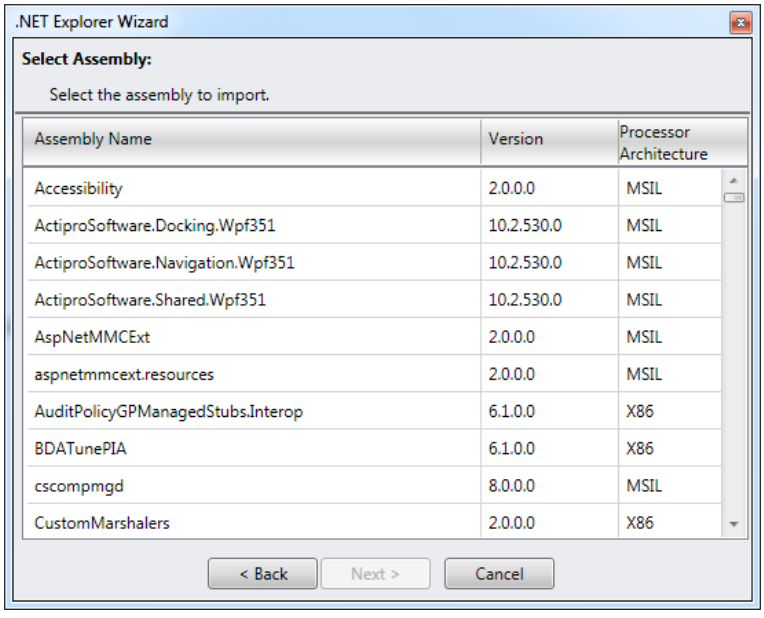


Abbildung 7: Auswahl der zu integrierenden Assembly

In diese Maske werden tabellarisch alle im global assembly cache verwalteten .Net Assemblies angezeigt.

MD Consulting & Informationsdienste GmbH

Wie aus der Dokumentation (sieheAbbildung 1: Die (Online) Dokumentation der Eigenschaft MyDocuments,Seite 5) hervorgeht, soll die Assembly Microsoft.VisualBasic in die SQLWindow-Anwendung integriert werden.

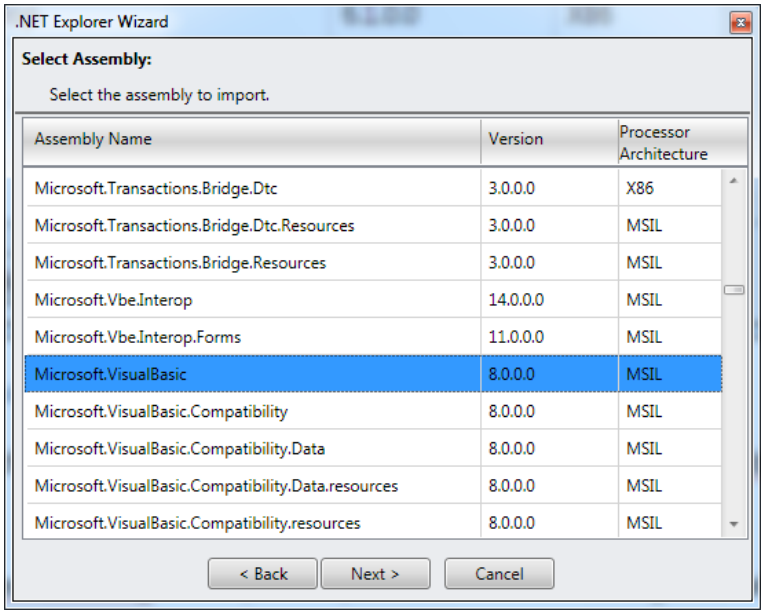


Abbildung 8: Auswahl der Microsoft.VisualBasic Assembly

Nach dem Anklicken der Schaltfläche Next werden die Klassen, die in der auswählten .Net Assembly organisiert sind, angezeigt.

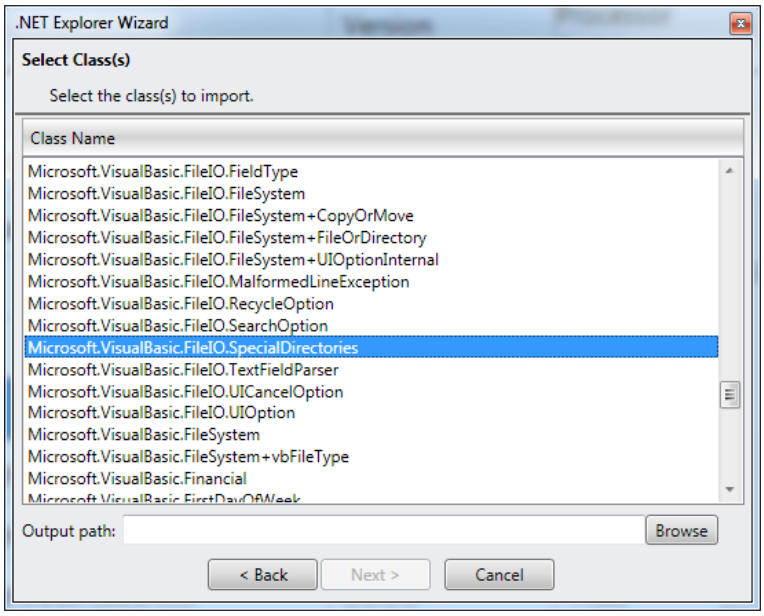


Abbildung 9: Auswahl der Klasse „SpecialDirectories“

In dieser Auswahlmaske kann eine Klasse oder mehrere Klassen aus einer .Net Assembly ausgewählt werden, für die die entsprechende Proxy-Funktionalität erstellt werden soll.

Im Feld Output Path muss noch das Verzeichnis angegeben werden, in dem die zu erstellende Bibliotheks-Datei abgelegt werden soll. (Wir hatten vorher ein Verzeichnis mit dem Namen Beispiel angelegt, in dem schon die Hauptdatei abgespeichert worden war.)

MD Consulting & Informationsdienste GmbH

Nach Auswahl einer oder mehrerer Klassen und der Definition des Ausgabe-Verzeichnisses wird die Schaltfläche Next schwarz gesteuert – jetzt kann die Erstellung der Bibliotheksdatei aufgerufen werden.

Die Bibliotheksdatei mit dem Namen Microsoft.VisualBasic wird im spezifizierten Verzeichnis angelegt und der Verweis auf diese Datei wird im Bereich File include: der Quellcodedatei eingebunden.

Die erzeugte Bibliotheksdatei Microsoft.VisualBasic.apl

Die mit Hilfe des .Net Explorers erstellte Datei sollte nicht modifiziert werden, wie aus der Bemerkung im Description-Teil der Bibliothek hervorgeht („GAIL Proxy - Generated by TD 6.1. Direct modification of these classes by hand is strongly discouraged.")

Allerdings ist es interessant, sich den Aufbau und die Bestandteile der Bibliotheksdatei genauer anzuschauen:

- Als External Functions wird die GailWrapper.dll automatisch eingebunden. In dieser DLL sind alle Funktionen gekapselt, mit denen die Schnittstelle zwischen SQLWindows und der entsprechenden .Net Assembly abgebildet wird.
- Im Bereich Variables sind zwei Variablen definiert, mit denen eventuell auftretende Fehler bei der Kommunikation zwischen SQLWindows-Anwendung und .Net Assembly behandelt werden können. Die Variable bGailDisableAutoReportErrors wird auf TRUE gesetzt, wenn mögliche Fehler in der Anwendung selbst behandelt werden sollen. Um das entsprechend sinnvoll vornehmen zu können, kann auf die Fehlernummer des aufgetretenen Fehlers nGailErrorMsg zurückgegriffen werden.
- Im Bereich Internal Functions sind alle Eigenschaften - sie können nur abgefragt, aber nicht gesetzt werden - aus der Klasse SpecialDirectories, unter anderem auch die für die Beispielanwendung benötigte Eigenschaft Microsoft_VisualBasic_FileIO_SpecialDirectories_get_MyDocuments () als Funktionen hinterlegt.

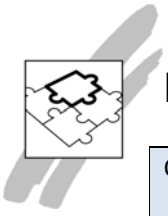
Es sind daher in der Bibliotheksdatei alle Eigenschaften der Spezialverzeichnisse abgebildet, sodass nicht nur Meine Dokumente, sondern auch die physischen Pfade zu logischen Verzeichnisse wie z.B. Meine Bilder potentiell abgefragt werden können.

Beispielanwendung – Kodierung des Tree View Controls

Nachfolgend wird der für dieses Beispiel maßgebliche Ausschnitt der Bedienoberfläche angezeigt.

Abbildung 10: Tree View der Beispielanwendung (Win32)

Bei Tree View Control sind folgende Aktionen hinterlegt:



MD Consulting & Informationsdienste GmbH

```
On SAM_Create
   Call MeineVerzeichnisseAnzeigen( hWndItem )
On SAM_ItemSelected
   Call VerzeichnisPrüfenUndAnlegen( hWndItem )
```

Source Code 1: Behandlung von Nachrichten bei Tree View Control

Bei Erstellen des Tree View Controls wird die Funktion MeineVerzeichnisseAnzeigen () ausgeführt. In dieser Funktion wird der physische Pfad des logischen Verzeichnisses Meine Dokumente ermittelt und weitere vordefinierte Unterverzeichnisse angezeigt. Wenn die Unterverzeichnisse bereits existieren, werden die Bezeichnungen mit Normalschrift angezeigt. Wenn sie noch nicht angelegt wurden, werden die Namen der Unterverzeichnisse grau angezeigt.

Wird ein (Unter)Verzeichnisname im Tree View angeklickt, wird geprüft, ob das Unterverzeichnis bereits existiert. Falls das nicht der Fall ist, wird es unter dem Verzeichnis Meine Dokumente angelegt.

```
Function: MeineVerzeichnisseAnzeigen
   Description:
   Returns
   Parameters
      Window Handle: p_hWnd
   Static Variables
   Local variables
      String: sMeineDokumente
      String: sMeineVerzeichnisse[*]
      Number: nWurzel
      Number: nKnoten
      Number: n
      String: sMeinSpeziellerPfad
   Actions
      Set sMeineVerzeichnisse[0] = 'Export - Meine Angebote'
      Set sMeineVerzeichnisse[1] = 'Export - Meine Auftragsbestätigungen'
      Set sMeineVerzeichnisse[2] = 'Export - Meine Bestellungen'
      Set sMeineVerzeichnisse[3] = 'Export - Meine Rechnungen'
      Set sMeineVerzeichnisse[4] = 'Export - Meine Zahlungseingänge'
      Set sMeineVerzeichnisse[5] = 'Export - Meine sonstigen Export-Dateien'
      Set sMeineDokumente = Mi-
crosoft_VisualBasic_FileIO_SpecialDirectories_get_MyDocuments ()
      Set nWurzel = SalTreeGetFirstItem( p_hWnd, 0 )
      Call SalTreeSetItemTooltip( p_hWnd, nWurzel, sMeineDokumente )
      Set n = 0
      While n <= 5
         Set nKnoten = SalTreeInsertItem( p_hWnd, nWurzel, -1, sMeineVerzeichnisse[n] )
         Set sMeinSpeziellerPfad = sMeineDokumente || '\\' || sMeineVerzeichnisse[n]
         Call SalTreeSetItemTooltip( p_hWnd, nKnoten, sMeinSpeziellerPfad )
         If SalFileSetCurrentDirectory( sMeinSpeziellerPfad )
            Call SalTreeSetItemTextColor( p_hWnd, nKnoten, COLOR_Default )
         Else
            Call SalTreeSetItemTextColor( p_hWnd, nKnoten, COLOR_Gray )
         Set n=n+1
      Call SalTreeExpandItem( p_hWnd, nWurzel )
```

Source Code 2: Die Funktion MeineVerzeichnisseAnzeigen

MD Consulting & Informationsdienste GmbH

Im Beispielcode wird lediglich eine Eigenschaft aus dem .Net-Framework abgefragt, um den physischen Namen des logischen Verzeichnisses „Meine Dokumente“ auf dem Rechner, auf dem die Anwendung läuft, zu ermitteln.

Die Funktion VerzeichnisPrüfenUndAnlegen wird aufgerufen, wenn der Benutzer ein (eigenes) Verzeichnis anklickt. In diesem Beispielcode wird zunächst geprüft, ob das „logische“ Verzeichnis bereits physisch existiert. Sollte das nicht der Fall sein, muss es angelegt werden.

```
Function: VerzeichnisPrüfenUndAnlegen
   Description:
   Returns
   Parameters
      Window Handle: p_hWnd
   Static Variables
   Local variables
      Boolean: bOk
      String: sVerzeichnis
      Number: nKnoten
   Actions
      Set nKnoten = SalTreeGetSelectedItem( p_hWnd )
      Set sVerzeichnis = SalTreeGetItemTooltip( p_hWnd, nKnoten)
      If NOT SalFileSetCurrentDirectory( sVerzeichnis )
         Set bOk = SalFileCreateDirectory( sVerzeichnis )
         If bOk
            Set bOk = SalTreeSetItemTextColor( p_hWnd, nKnoten, COLOR_Default )
      Return bOk
```

Source Code 3: Die Funktion VerzeichnisPrüfenUndAnlegen

In diesem Beispielcode wird keine Funktion aus dem .Net-Framework aufgerufen – es wird mit dem Standard-Sprachumfang der SAL-Sprache gearbeitet.

Beispielanwendung WPF

Um die Beispielanwendung als WPF-Anwendung ausführen zu können, müssen auf der Basis des vorliegenden Codes folgende Schritte ausgeführt werden:

1. Änderung des Compilers von Win32 auf WPF Desktop
2. Änderung von File Include zu Symbol File Include
3. Ausführen der Anwendung

Änderung des Compilers

Durch Aufruf der Menüfunktion Project, Build Settings ... wird der Dialog zur Definition des Compilers bzw. der Kompilierungsart aufgerufen.

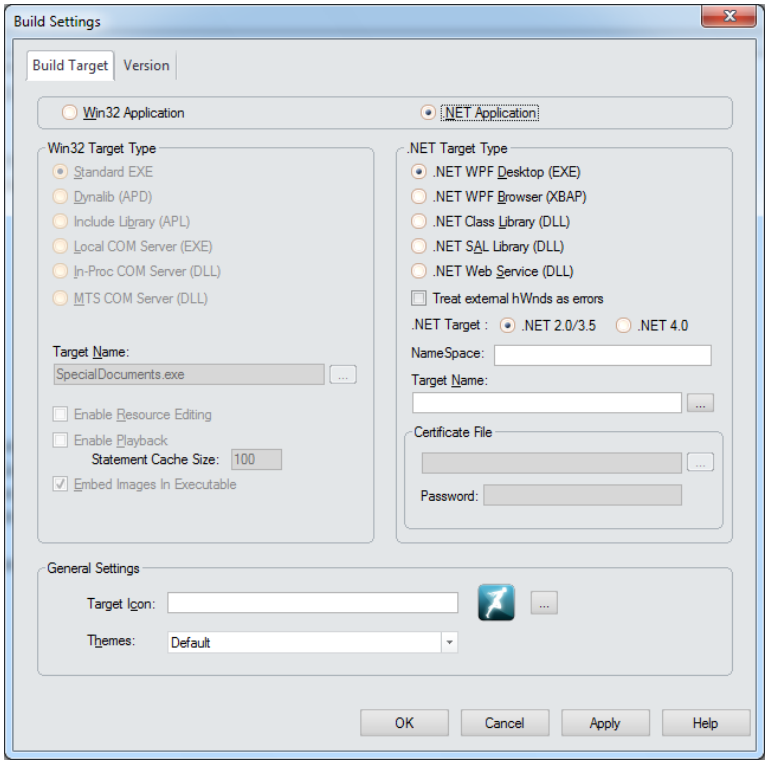


Abbildung 11: Veränderung des Compilers für den vorliegenden Quellcode

In diesem Fall wird lediglich die Option von Win32 Standard EXE auf .NET WPF Desktop geändert und der Dialog mit Ok beendet. (Der eventuell angezeigte Hinweis, eine Fehlerdatei zu erstellen, kann abgebrochen werden.)

Änderung File Include zu Symbol File Include

Im nächsten Schritt wird anstelle der APL-Bibliotheksdatei die entsprechende Symbol-Datei als Library eingebunden.

```
Libraries
   Symbol File Include: Microsoft.VisualBasic.axl
```

Source Code 4: Einbinden der Symboldatei

Die Datei Microsoft.VisualBasic.axl wurde bereits vom .Net Explorer-Assistenten angelegt, als wir die Win32 Bibliotheksdatei angelegt hatten. Bei einer Win32-Applikation muss die Schnittstelle zur .Net Assembly noch über den GAILWrapper abgebildet werden, was bei einer .NET WPF-Anwendung nicht benötigt wird: eine WPF-Anwendung benötigt keine Wrapper-Funktionalität.

Bei einem Blick auf die Code-Sektion External Assemblies fällt nun auf, dass innerhalb der Microsoft.VisualBasic-Assembly auf zwei Komponenten verwiesen wird: VisualBasic und mscorlib.

Innerhalb der Komponente VisualBasic befindet sich eine Globals- und eine Class-Sektion. Innerhalb der globalen Sektion befinden sich die Funktionen, mit denen die physischen Pfade der „logischen" Verzeichnisse zurückgegeben werden. Da die Funktionen im globalen Bereich übernommen werden, brauchen sie bei der Codierung auch nicht instantiiert zu werden: globale Funktionen einer .Net-Klasse sind global innerhalb von SQLWindows aufrufbar.

Aus diesem Grund kann der bestehende Source Code nicht modifiziert als WPF-Anwendung kompiliert und ausgeführt werden.

Abbildung 12: Die Beispielanwendung als WPF-Anwendung

Im Tree View Control werden die von der Anwendung verwalteten Verzeichnisse angezeigt: beim Wurzelknoten handelt es sich um den logischen Namen des speziellen Verzeichnisses für „Dokumente“. Darunter werden die logischen Namen der Verzeichnisse angezeigt, die durch die Anwendung verwaltet werden.

Zusammenfassung

Die Unterschiede zwischen Win32 und WPF-Anwendung hinsichtlich der Codierung sind nicht gravierend. Das ist wichtig, wenn geplant ist, mittelfristig aus einer Win32-Anwendung insgesamt eine WPF-Anwendung zu machen. Während die Win32-Anwendung über eine InterOp-Technologie mit den entsprechenden .Net-Assemblies kommuniziert, fällt bei einer WPF-Anwendung lediglich dieser „Zwischenschritt“ weg.

Um den Sourcecode einer Win32-Anwendung für die Kompilierung als WPF-Anwendung vorzubereiten, muss also lediglich die Art der Einbindung der Proxy-Datei geändert werden: bei einer Win32-Anwendung wird eine APL-Datei eingebunden, in der der GAIL-Wrapper organisiert ist. Über die GAIL-Wrapper-Funktionen wird anhand der Informationen in der Symbol-Datei mit der entsprechenden Klasse in der .Net Assembly kommuniziert.

In einer WPF-Anwendung dagegen wird über Symbol File Include lediglich die entsprechende Symbol-Datei (, die ja bereits mit dem .Net Explorer erzeugt worden ist), in den Quellcode der Beispielanwendung integriert.

Obwohl die Proxy-Funktionalitäten bei Win32- und WPF-Anwendungen teilweise unterschiedlich organisiert sind, braucht aufgrund des implementierten Scopes keine Veränderung am eigenen Code vorgenommen zu werden: die Integration von .Net-Framework-Funktionalitäten ist daher „aufwärts kompatibel“.

MD Consulting & Informationsdienste GmbH

Aufgabenstellung 2: Überprüfung einer Eingabe (String)

Eine Eingabe soll validiert werden. Eigentlich kein Problem, da SQLWindows bereits eine Vielzahl von Funktionen zur Validierung bereitstellt. Allerdings bietet das .Net Framework eine sehr umfassende Technologie zur Validierung von (Eingabe-) Strings an: die regulären Ausdrücke (regular expressions). Diese Technologie bietet umfassende Möglichkeiten an, um prüfen zu können, ob Zeichenketten (Strings) einen bestimmten, formalen Aufbau haben, die Validierungen ermöglichen, die weit über den Sprachumfang von SAL-Funktionen hinausgehen.

Lösungsweg

Leider ist die Technologie der regulären Ausdrücke so umfangreich, dass schon ein sehr einfaches Beispiel zunächst recht kompliziert aussehen kann. Als erstes Beispiel sollen (in Deutschland übliche) Schulnoten dienen, d.h. es sind lediglich die Zeichen 1,2,3,4,5 oder 6 als Eingabe zulässig. Wie sieht die Validierung einer Eingabe als Schulnote aus, wenn wir auf eine Validierung mithilfe von SAL-Funktionen verzichten wollen (oder müssen)?

Die Technologie der regulären Ausdrücke enthält eine Beschreibungssprache mit der beliebige Zeichenketten formal beschrieben werden können. Diese Muster werden verwendet, um sie mit einer realen Zeichenkette zu vergleichen – genauer: um festzustellen, ob die gegebene Zeichenkette der formalen Beschreibung entspricht.

Klassische deutsche Schulnoten können, wie gesagt, nur aus den Zeichen 1 bis 6 bestehen. Als Beschreibung dieser Bedingung als regulärer Ausdruck könnte daher zunächst [123456] gewählt werden. Die Kurzform dieser Beschreibung lautet [1-6].

Die Beschreibung ist damit noch nicht abgeschlossen, da beispielsweise auch die Zeichenkette 11 nur aus Zeichen besteht, die zulässig sind. Es muss daher auch die Anzahl der Zeichen auf einstellig begrenzt werden. Dieses geschieht in der Syntax der regulären Ausdrücke durch geschweifte Klammer {}. Daher lautet die Musterbeschreibung nunmehr [1-6]{1}.

Die Musterbeschreibung soll sich auf die gesamte Zeichenkette beziehen. Eine Zeichenkette als regulärer Ausdruck wird eingeleitet mit dem ^-Zeichen und beendet mit dem $-Zeichen. Daher lautet die Beschreibung klassischer deutscher Schulnoten als regulärer Ausdruck nunmehr ^[1-6]{1}$.

Die Methode, um eine gegebene Zeichenkette mit einer Beschreibung zu vergleichen, heißt isMatch (). Sie ist im .Net Framework in der Regex-Klasse im Namensraum System.Text.RegularExpressions hinterlegt.

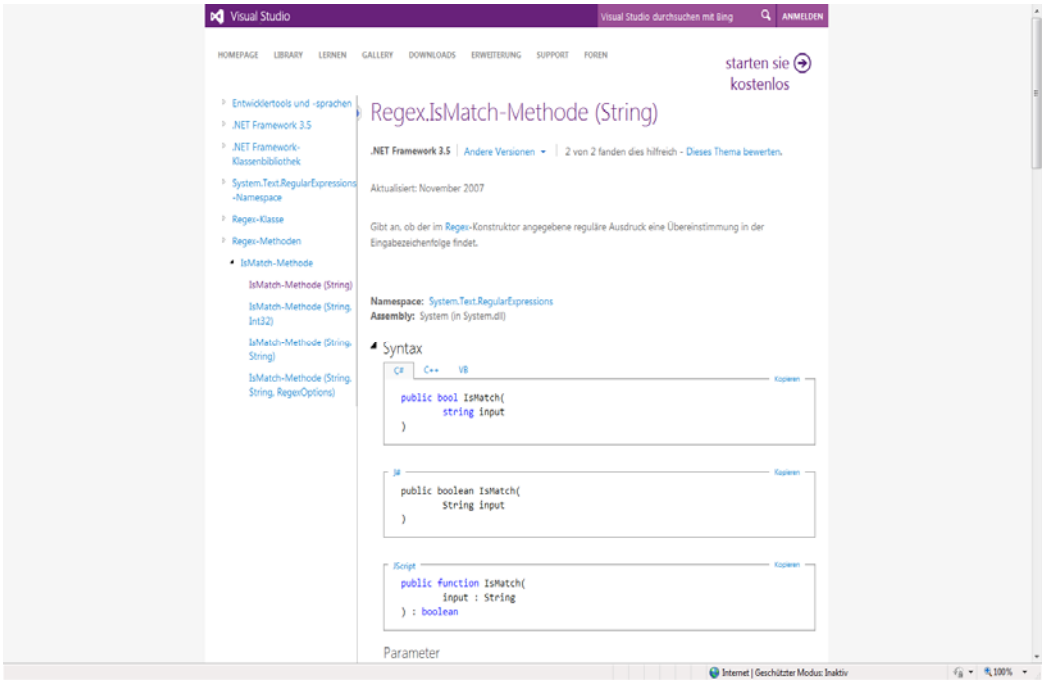


Abbildung 13: Dokumentation der IsMatch-Methode

Beispielanwendung 2: Schulnote

Die Oberfläche der Beispielanwendung sieht folgendermaßen aus:

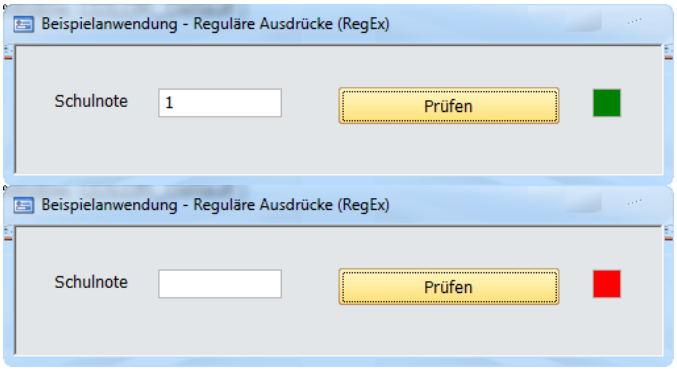


Abbildung 14: Oberfläche der Beispielanwendung (Win32)

Im oberen Beispiel wurde der Wert 1 überprüft – es handelt sich um eine gültige Schulnote. Im unteren Beispiel wurde nichts eingegeben – es handelt sich um eine ungültige Schulnote.

Um sich die Proxy-Funktionalität für den Zugang zur RegEx-Klasse zu erzeugen, muss der .Net Explorer aufgerufen werden und die Assembly mit dem Namen System ausgewählt werden.

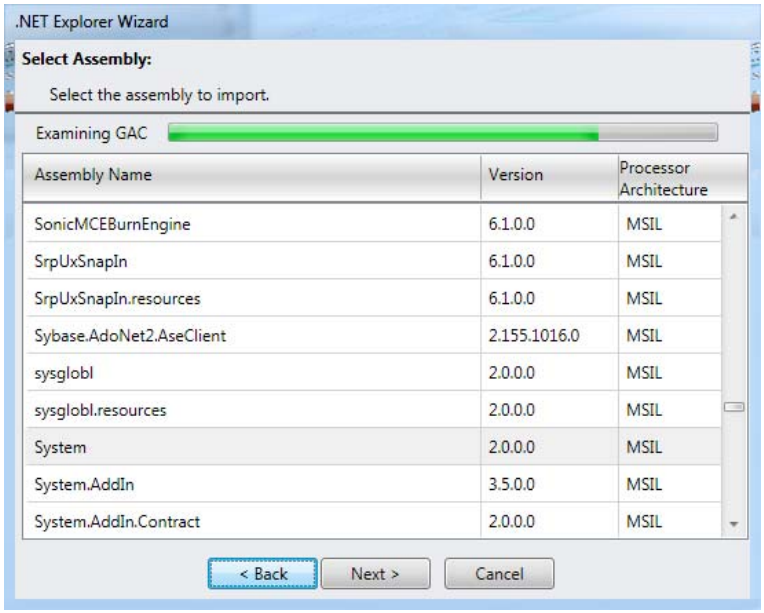


Abbildung 15: Auswahl der Assembly System

Aus dieser Assembly wird lediglich die Klasse Regex ausgewählt und die entsprechende Bibliothekdatei (einschließlich der entsprechenden Symboldatei) mit dem .Net Explorer erstellt, abgespeichert und in den Quelltext der vorhandenen Anwendung eingebunden.

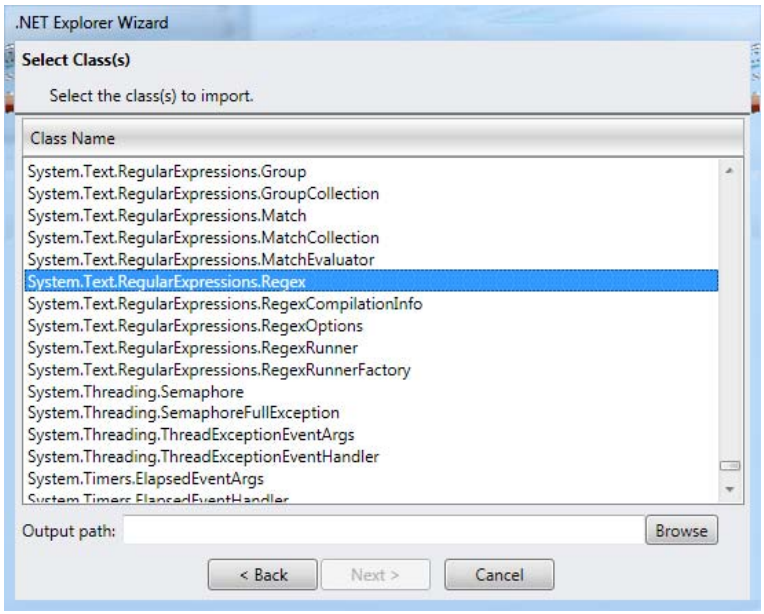


Abbildung 16: Auswahl der Regex-Klasse

Die Codierung des Aufrufs ist einfach: nach dem Klick auf dier Schaltfläche Prüfen wird die Funktion EingabePrüfen des Form Windows ausgeführt. Die Funktion EingabePrüfen () hat folgenden Aufbau:

```
Function: EingabePrüfen
…
   Local variables
   Actions
      If System_Text_RegularExpressions_Regex_IsMatch( df1, '^[1-6]{1}$' )
         Call SalColorSet( df2, COLOR_IndexWindow, COLOR_DarkGreen )
      Else
         Call SalColorSet( df2, COLOR_IndexWindow, COLOR_Red )
```

Source Code 5: Die Funktion EingabePrüfen

Mit Hilfe der internen (Win32-Variante) oder globalen (WPF-Variante) Funktion System_Text_RegularExpressions_Regex_IsMatch () wird die im Datenfeld df1 stehende Zeichenkette mit dem regulären Ausdruck ^[1-6]{1}$ geprüft. Der Rückgabewert dieser Funktion gibt an, ob die eingegebene Zeichenkette der Beschreibung des regulären Ausdrucks entspricht.

Die (zum .Net Framework gehörende) Methode IsMatch ist geeignet, wenn lediglich geprüft werden soll, ob eine gegebene Zeichenkette der als regulärer Ausdruck formulierten Bedingung entspricht (oder nicht). In vielen Fällen ist es allerdings notwendig, weitergehende Informationen darüber zu erhalten, an welcher Stelle eine gegebene Bedingung eingehalten wird (oder nicht).

Die Vorgehensweise soll anhand einer einfachen Buchstabenprüfung demonstriert werden.

Beispielanwendung 2a – Matches

In diesem Beispiel soll eine einfache Suchfunktion implementiert werden. Die nachfolgende Abbildung zeigt die Bedienoberfläche der Anwendung.

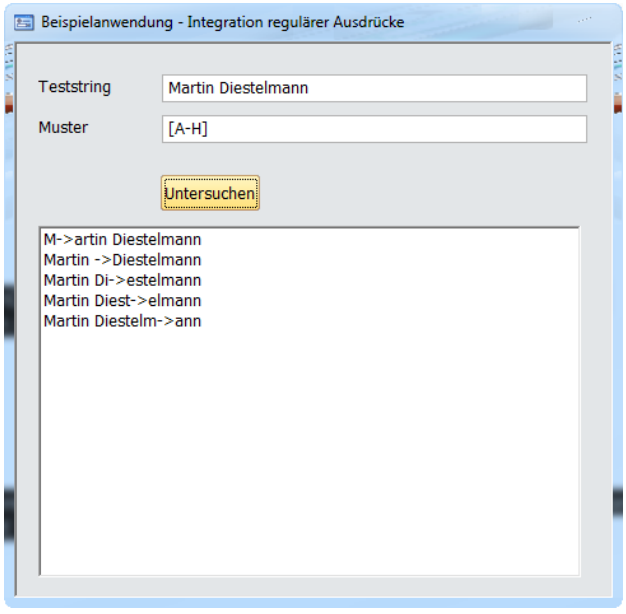


Abbildung 17: Die Oberfläche der Beispielanwendung

Im ersten Datenfeld steht eine beliebige Zeichenkette, während im zweiten Datenfeld das entsprechende Vergleichsmuster (als einfacher regulärer Ausdruck) hinterlegt wird. Nach der Definition im Eingabefeld sind lediglich die Zeichen A bis H groß geschrieben in dem zu vergleichenden Muster zulässig. Allerdings – das wird aus dem unten stehenden Quellcode deutlich – wird der Mustervergleich unabhängig von Groß- bzw. Kleinschreibung durchgeführt.

MD Consulting & Informationsdienste GmbH

Nach Anklicken der Schaltfläche Untersuchen wird der Teststring nach Vorkommen des in Muster hinterlegten Ausdrucks untersucht und die Ergebnisse werden in der List Box angezeigt.

In diesem Bespiel sind lediglich die Zeichen A bis H zulässig und das Vorkommen dieser Zeichen – das heißt: die Verletzung der Regel – wird in der Listbox mit dem Zeichen „-->“ verdeutlicht.

Codierung

Um genauere Informationen über die Stellen in einer Zeichenkette zu erhalten, die eine als regulärer Ausdruck definierte Bedingung erfüllen (oder nicht), muss die Methode Matches verwendet werden.

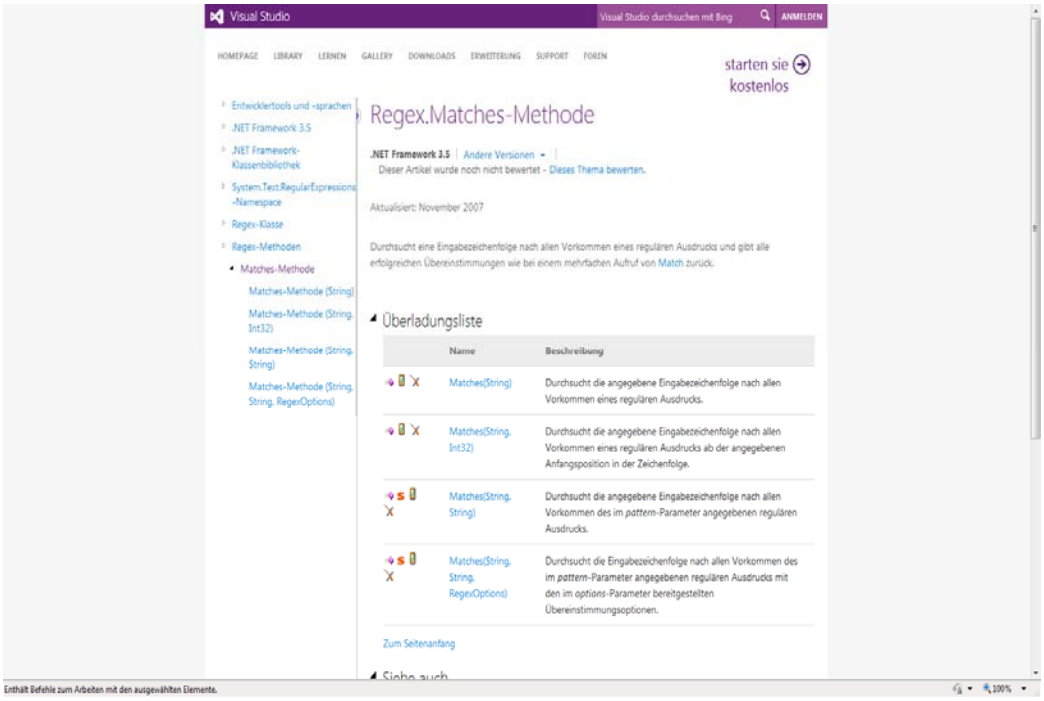
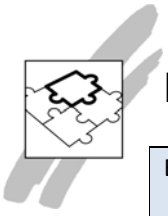


Abbildung 18: Dokumentation der Methode Matches in der Klasse RegEx

Im .Net Framework ist die Methode „überladen“ definiert, d.h. sie kann einer unterschiedlichen Anzahl Parametern, wie der abgebildeten Dokumentation zu entnehmen ist, aufgerufen werden. Eine derartige Überladung gibt es in SQLWindows nicht. Daher werden bei der Generierung des Proxies für „überladene“ Methoden mehrere Proxy-Funktionen (mit der Kennung Overload[n]) erzeugt.

Um die Methode Matches aus dem .Net-Framework in einer SQLWindows-Anwendung verwenden zu können, wird daher im vorliegenden Fall die Funktion System_Text_RegularExpressions_Regex_Matches_Overload2 verwendet. Diese Funktion hat in SQLWindows drei Parameter: die Eingabezeichenkette, den regulären Ausdruck und eine numerische Option.

Die zulässigen numerischen Optionen eines Parameters werden durch den Proxy-Generator von SQLWindows gegebenenfalls als Enumerations übernommen.



MD Consulting & Informationsdienste GmbH

```
Enum: System_Text_RegularExpressions_RegexOptions
   Item: None = 0
   Item: IgnoreCase = 1
   Item: Multiline = 2
   Item: ExplicitCapture = 4
   Item: Compiled = 8
   Item: Singleline = 16
   Item: IgnorePatternWhitespace = 32
   Item: RightToLeft = 64
   Item: ECMAScript = 256
   Item: CultureInvariant = 512
```

Source Code 6: Die möglichen Parameterwerte der Option in der überladenen Methode Matches

Die Überprüfung auf Vorkommen des Musters in der gegebenen Zeichenkette erfolgt in der Funktion TestMatches.

```
Function: TestMatches
   Description:
   Returns
   Parameters
   Static Variables
   Local variables
      System_Text_RegularExpressions_MatchCollection: Ergebnis
      System_Text_RegularExpressions_Match: einTreffer
      Number: nAnzahlTreffer
      Number: n
      Number: nPosition
      String: sAnzeige
   Actions
      Call SalListClear( lb1 )
      Set Ergebnis = System_Text_RegularExpressions_Regex_Matches_Overload2( df1,df2, Sys-
tem_Text_RegularExpressions_RegexOptions.IgnoreCase )
      Set nAnzahlTreffer = Ergebnis.get_Count(  )
      If nAnzahlTreffer > 0
         Set n = 0
         While n < nAnzahlTreffer
            Set einTreffer = Ergebnis.get_Item( n )
            Set nPosition = einTreffer.get_Index(  )
            Call SalStrReplace ( df1, nPosition,0, '->', sAnzeige)
            Call SalListAdd ( lb1, sAnzeige)
            Set n=n+1
```

Source Code 7: Die Funktion TestMatches

Im vorliegenden Code wird die Methode Matches mit drei Parametern (Overload2) aus der Regex-Klasse aufgerufen: als erster Parameter wird der Vergleichsstring eingegeben, als zweiter Parameter der regulärer Ausdruck selbst und als dritter Parameter wird über einen Enumerationswert spezifiziert, dass Groß- und Kleinschreibung beim Vergleich unberücksichtigt bleiben sollen. Der Rückgabewert der Methode Matches ist ein MatchCollection-Objekt mit dem Namen Ergebnis. In dem MatchCollection-Objekt wiederum befinden sich beliebig viele Objekte vom Typ Match (einTreffer).

In der While-Schleife werden die einzelnen Treffer durchgearbeitet, um die Stellen im String hervorheben zu können, wo die als regulärer Ausdruck an gegebenen Bedingungen zutreffen. Die für diesen „Job“ notwendigen Einzelfunktionen werden

von den entsprechenden Klassen (MatchCollection, Match) entsprechend zur Verfügung gestellt und im SQLWindows Source Code lediglich aufgerufen.

Beispielanwendung 3: Arbeiten mit XML-Dokumenten

XML-Dokumente und –Strukturen eignen sich besonders zum plattformübergreifenden Datenaustausch, da sich formal standardisiert sind. Aus diesem Grund gibt es die auf den unterschiedlichsten Plattformen Hilfsmittel (XML-Leser, Parser, usw.), mit denen die Verarbeitung von teilweise komplexen Strukturen vereinfacht wird.

Auch im .Net Framework ist eine Vielzahl unterschiedlicher Hilfsmittel für die Verarbeitung von XML-Strukturen integriert. Anhand eines einfachen Beispiels soll in einer WPF-Anwendung gezeigt werden, wie ein XML-Dokument (mit Daten aus der Datenbank) einfach erstellt werden kann.

Sinn und Zweck der Beispielanwendung besteht in der Erstellung eines XML-Dokuments. In diesem Beispiel wird das XML-Dokument automatisch (ohne Benutzereingriff) nach dem Start der Anwendung erstellt. Deshalb entfällt in diesem Beispiel die Darstellung der Bedienoberfläche der Beispielanwendung.

Lösungsweg

Die unterschiedlichsten Hilfsmittel für die Verarbeitung von XML-Dokumenten sind im .Net Framework in der Klasse XMLDocument organisiert. Die nachfolgend dargestellte Webseite zeigt den Eingang zur Online-Dokumentation des Namensraums System.XML.

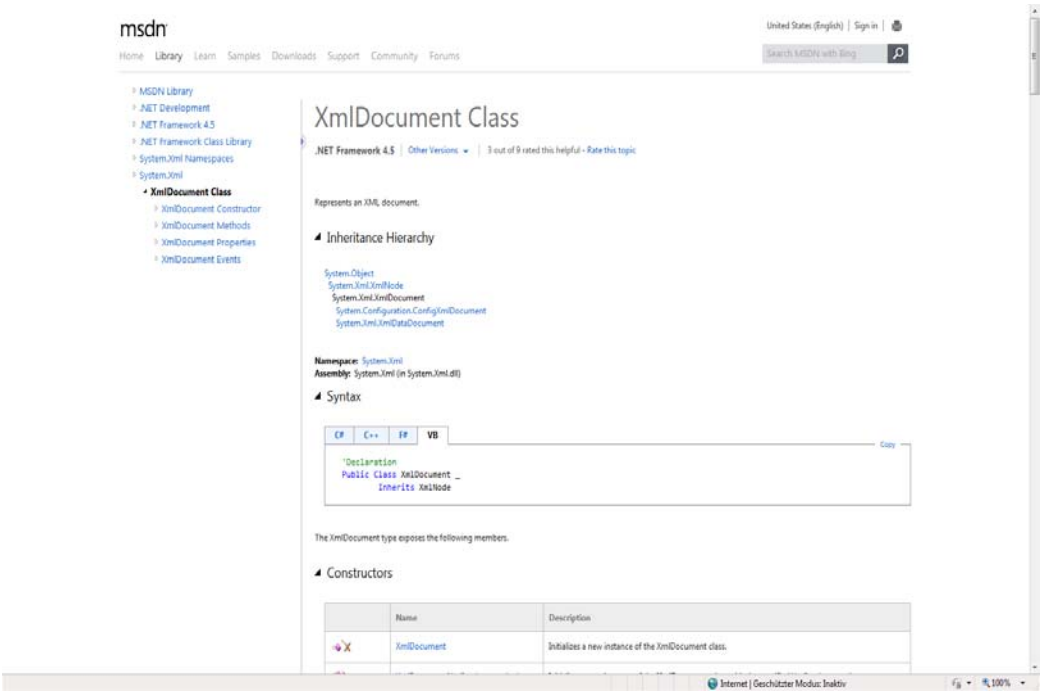


Abbildung 19: Dokumentation XMLDocument

Der Dokumentation kann folgendes entnommen werden:

- Die „Hilfsmittel“ befinden sich im Namensraum System.XML.
- Innerhalb dieses Namensraums befindet sich eine Klasse XMLDocument(, die von der Klasse XMLNode angeleitet ist.

Mit diesen Informationen können wir die notwendige Proxy-Datei für die Verarbeitung von XML-Strukturen in einer SQLWindows WPF-Anwendung erstellen.

Der .Net Explorer wird aus der Entwicklungsumgebung gestartet.

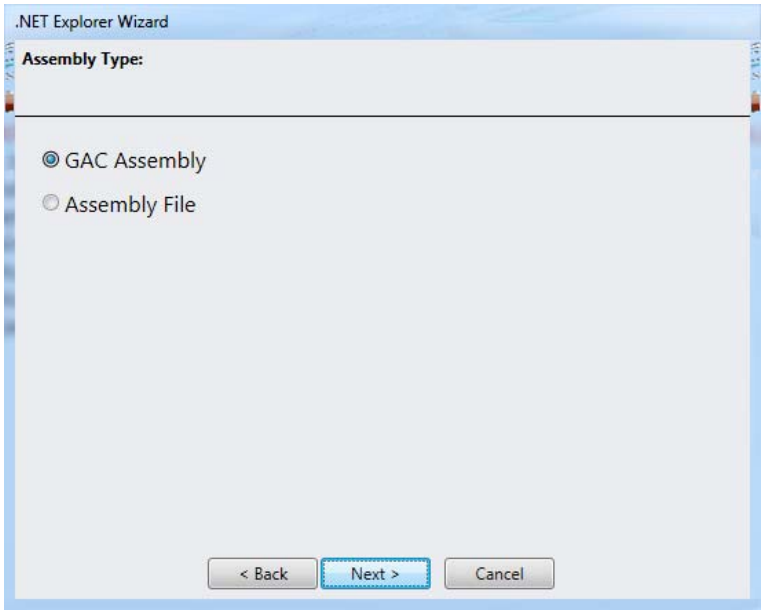


Abbildung 20: Die Assembly Sytem.XML wird im GAC verwaltet

Da die auszuwählende Assembly im global assembly cache verwaltet wird, wird diese Option beibehalten. Mit Next gelangt man in den nächsten Auswahlbildschirm.

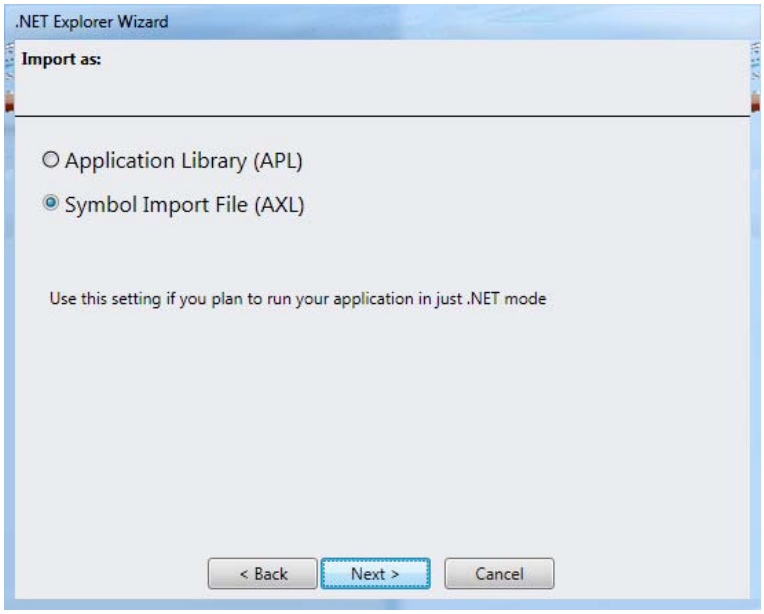


Abbildung 21: Bereitstellung der innerhalb einer WPF-Anwendung

Da die zu erstellende Anwendung eine WPF-Anwendung werden soll, wird lediglich die Symbol-Datei durch den .Net Assistenten erstellt. Mit Next gelangt man in den nächsten Auswahlbildschirm.

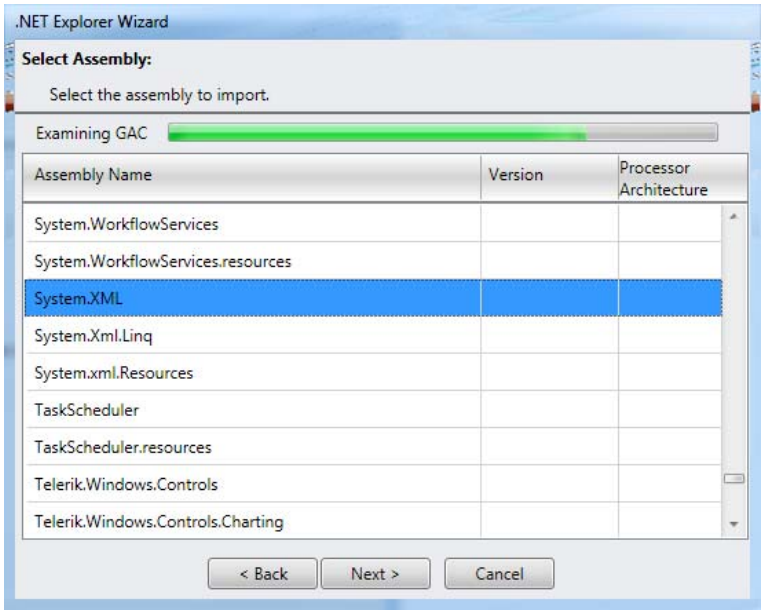


Abbildung 22: Auswahl der Assembly System.XML

Da ein XML-Dokument erstellt werden soll, wird die Assembly System.XML ausgewählt. Mit Next gelangt man zum nächsten Auswahlbildschirm.

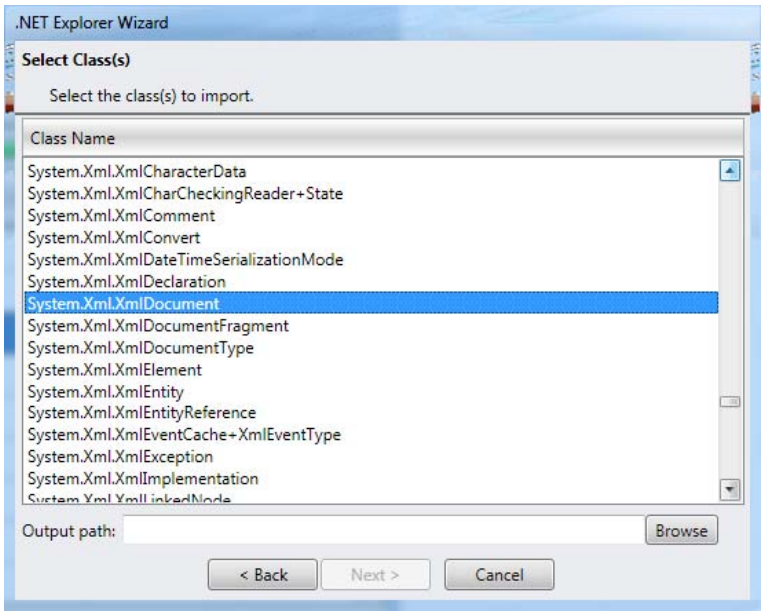


Abbildung 23: Auswahl der Klasse XMLDocument

Es wird lediglich die Klasse XMLDocument ausgewählt. Der .Net Assistent generiert Proxy-Funktionalitäten zu allen von der Klasse XMLDocument abhängigen Komponenten.

Nach der Angabe eines Verzeichnisses, in das die zu erstellende Symbol-Datei geschrieben werden soll, wird die Next-Schaltfläche schwarz gesteuert angezeigt: die Symbol-Datei wird erstellt.

MD Consulting & Informationsdienste GmbH

Implementierung in einer WPF-Anwendung

In der Beispielanwendung sollen die Daten aus der Tabelle COMPANY in der Datenbank ISLAND in ein XML-Dokument geschrieben werden, das als Datei unter dem Namen „Elemente und Attribute.xml“ im Dateisystem abgespeichert werden soll.

Der schematische Aufbau der zu schreibenden XML-Struktur sieht folgendermaßen aus:

- Der (nur einmal auftretende) Wurzelknoten der hierarchischen Struktur erhält den Namen Ergebnismenge und keine eigene „Beschriftung“
- Der Knoten Firma kommt so oft vor, wie es Datensätze in der Ergebnismenge gibt. Jeder Knoten Firma ist ein Kindknoten des Wurzelknotens Ergebnismenge. Die Beschriftung eines jeden Knotens Firma ist der tatsächliche Name (COMPANY_Name) der entsprechenden Firma. Mit dem Anlegen eines Knotens Firma soll gezeigt werden, wie XML-Elemente mithilfe der XML-Klasse erzeugt und in ein XML-Dokument eingefügt werden können.
- In diesem Beispiel sind die Knoten Stadt und Land als Attribute des übergeordneten Knotens definiert, damit mit der Beispielanwendung gezeigt werden kann, wie man Attribute (eines Elements) definieren kann.

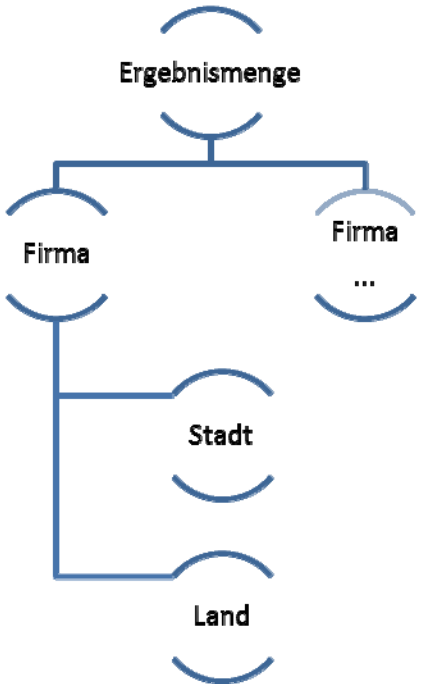
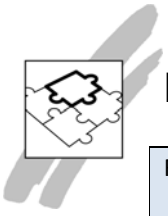


Abbildung 24: Visualisierung der Struktur des XML-Dokuments

Mit der Funktion XMLDokumentErstellen wird ein XML-Dokument erstellt, die entsprechende Struktur aufgebaut und am Ende unter dem Namen „Elemente und Attribute.xml“ im Verzeichnis der laufenden Anwendung abgespeichert.



MD Consulting & Informationsdienste GmbH

```
Function: XMLDokumentErstellen
  Description:
  Returns
  Parameters
  Static Variables
  Local variables
    System_Xml_XmlDocument: doc
    System_Xml_XmlDeclaration: decl
    System_Xml_XmlElement: elem
    System_Xml_XmlElement: NewElem
  Actions
    Call doc.XmlDocument_Overload1()
    Set decl = doc.CreateXmlDeclaration( '1.0', 'UTF-8', STRING_Null )
    Call doc.AppendChild (decl )
    Set elem = doc.CreateElement_Overload1 ('Ergebnismenge')
    Set NewElem = doc.AppendChild (elem)
    Call DatenEinlesen ( doc, NewElem )
    Call doc.Save_Overload1 ('Elemente und Attribute.xml')
```

Source Code 8: Die Funktion XMLDokumentErstellen

Als Variablen der Funktion werden die Klassen XMLDocument, XMLDeclaration (einmal) und XMLElement (zweimal) definiert. Die Instanz doc von XMLDocument wird benötigt, um überhaupt ein XML-Dokument angelegen, verwalten und bearbeiten zu können, die Instanz decl wird benötigt, um eine XML-Deklaration (welcher Zeichensatz wird verwendet?) in das Dokument einfügen zu können. Die Instanz elem wird benötigt, um den Wurzelknoten mit der Beschriftung Ergebnismenge anlegen und in das Dokument doc einfügen zu können, während das Element NewElem als Vaterknoten verwendet wird, um die Daten aus der SELECT-Anweisung (siehe unten) an eben diesen Vaterknoten „hängen“ zu können.

Grundsätzlich funktioniert die Verarbeitung von XML-Strukturen also so, dass unterschiedliche Komponenten (Deklaration, Elemente, Attribute) zunächst angelegt und gestaltet werden, bevor sie dann an einer bestimmten Stelle in die hierarchische Knoten-Struktur eines XML-Dokuments eingefügt werden.

Mit Aufruf der Proxy-Funktion doc.XMLDocument_Overload1 wird die Instant doc letztlich zu einem XML-Dokument initialisiert. Nach erfolgreicher Ausführung dieser Funktion die Variable doc initialisiert und es kann auf die Methoden und Eigenschaften des Objekts doc zugriffen werden.

Mit Ausführung der Funktion doc.CreateXMLDeclaration wird die Instanz decl entsprechend der Parameterwerte initialisiert. Diese Deklaration wird mit der Funktion doc. AppendChild in das XML-Dokument doc eingefügt.

Analog wird mit der Funktion doc.CreateElement_Overload1 die Instanz elem initialisiert und als Name des Elements wird Ergebnismenge festgelegt. Dieses Element wird ebenfalls in das XML-Dokument doc eingefügt, wobei der Rückgabewert der AppendChild-Funktion der Instanzvariablen NewElem zugewiesen wird. Der Wert dieser Instanzvariablen wird später dazu genutzt, um die einzelnen Knoten der Datensätze aus der Datenbank an diesen Knoten – an den Vaterknoten – zu hängen.

In der Funktion DatenEinlesen wird der SELECT-Zugriff auf die Datenbank ausgeführt und die Ergebnismenge wird in das XML-Dokument eingearbeitet.

Am Ende der Funktion wird das erstellte Dokument im Dateisystem mit der Funktion doc.Save_Overload1 abgespeichert.

Das Einfügen von Elementen mit Attributen in eine XML-Struktur wird in der Funktion DatenEinlesen behandelt.

MD Consulting & Informationsdienste GmbH

Als Parameter der Funktion wird das bisher nur rudimentär angelegte XML-Dokument und ein „Handle“ auf den Vaterknoten (mit dem Namen Ergebnismenge) mitgegeben.

```
Function: DatenEinlesen
  Description:
  Returns
  Parameters
    System_Xml_XmlDocument: p_doc
    System_Xml_XmlNode: p_wurzel
  Static Variables
  Local variables
    Sql Handle: hSql
    String: sStatement
    Number: nReturn
    String: sFirma
    String: sStadt
    String: sLand
    System_Xml_XmlElement: elem
    System_Xml_XmlAttribute: attr
  Actions
    If SqlConnect ( hSql )
      Set sStatement = 'SELECT COMPANY_NAME, CITY, COUNTRY
                        FROM COMPANY INTO :sFirma, :sStadt, :sLand'
      If SqlPrepareAndExecute ( hSql, sStatement )
        While SqlFetchNext ( hSql, nReturn )
          Set elem = p_doc.CreateElement_Overload1 ('Firma')
          Call elem.set_InnerText (sFirma)
          Call elem.SetAttribute_Overload1 ('Stadt', sStadt )
          Call elem.SetAttribute_Overload1 ('Land', sLand )
          Call p_wurzel.AppendChild (elem)
        If SqlDisconnect ( hSql )
```

Source Code 9: Die Funktion DatenEinlesen

Neben den Variablen, die für den Zugriff auf die Datenbank benötigt werden, gibt es nur noch XMLElement elem und XMLAttribute attr.

Die Into-Variablen der SELECT-Anweisung werden folgendermaßen verwendet:

- Es wird ein XML-Element mit dem Namen Firma angelegt und damit die Instanzvariable elem initialisiert. Als Beschriftung des Elements ist der Firmenname vorgesehen, der mit der Funktion set_InnerText zugewiesen wird.
- Das Element Firma erhält zwei Attribute mit den Bezeichnungen Stadt und Land – die Zuweisung der Beschriftung erfolgt mit der Funktion SetAttribute_Overload1, wobei die INTO-Variablen sStadt und sLand mithilfe der Parameter dieser Funktion zugewiesen werden.
- Mit der Funktion p_wurzel.AppendChild wird das Element (mit seinen Attributen) an den Vaterknoten (Ergebnismenge) angehängt.

Die nachfolgend dargestellte Abbildung zeigt die programmtechnisch erstellte Datei „Elemente und Attribute.xml“ in einem Browser.

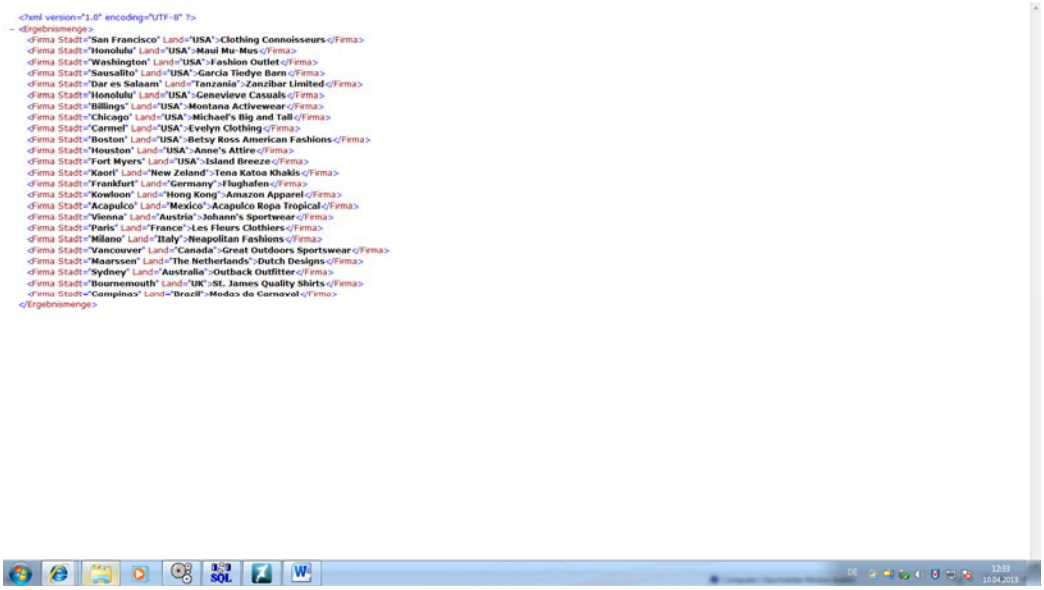


Abbildung 25: Die programmtechnisch erzeugte XML-Struktur (im Browser)

Zusammenfassung

In diesem Dokument konnte der tatsächliche Funktionsumfang des .Net-Frameworks lediglich angedeutet werden (Dateiorganisation, Validierung, XML). Es ging allerdings auch nicht um die Breite, sondern um die Frage, ob und wenn ja, wie die Funktionalitäten des .Net-Frameworks in SQLWindows-Anwendungen integriert werden können. Bei der Beantwortung dieser Frage war insbesondere von Interesse, ob die Integration von .Net Funktionalitäten in eine („klassische“) Win32-Anwendung eine mögliche Weiterentwicklung in Richtung einer originären .Net WPF-Anwendung behindern oder beeinträchtigen könnte. Die Antwort lautet nein, da bei einer veränderten Kompilierung einer Anwendung lediglich die Referenz (von File Include auf Symbol File Include) geändert werden müssen. Die auf diese Referenzen aufsetzende eigene Codierung kann unproblematisch auch mit dem neuen Compiler zur Ausführung gebracht werden.

Methoden und Eigenschaften: Bei der Proxy-Generierung mithilfe des .Net Explorers werden technische Unterschiede der Microsoft-Komponenten-Organisation nivelliert, da SQLWindows beispielsweise keine Unterschiede zwischen Methoden und Eigenschaften kennt. In SQLWindows wird alles als Function abgebildet, wobei die Namen der SQLWindows-Funktion durch das Präfix Set oder Get darauf hindeuten, dass es sich in der entsprechenden Klasse um Eigenschaften handelt.

Abbildung von Konstruktoren: Methoden (von Klassen) können im .Net-Umfeld mit Konstruktoren definiert sein. Dabei handelt es sich im Prinzip um automatisch auszuführende Funktionen bei der Initialisierung eines Objekts. In SQLWindows werden diese Konstruktoren nicht automatisch ausgeführt, sondern als Funktionen der jeweiligen Klasse als Proxy angeboten. Der SQLWindows-Entwickler muss die „Konstruktor-Funktion“ einer Methode explizit ausführen, indem er die entsprechende Funktion als Code ausführt. Ein Konstruktor wird als SQLWindows-Funktion mit dem Namenszusatz Overload versehen, wobei dieser Namenszusatz auch im Zusammenhang mit „Überladen“ verwendet wird.

Überladen: im .Net-Umfeld ist es möglich, eine Methode mit unterschiedlichen Parameterdefinitionen zu verwenden. Dieses „Überladen“ ist in SQLWindows nicht möglich. „Überladene“ Methode werden daher durch den Proxy-Generator in eine unterschiedliche Anzahl gleichnamiger Funktionen (mit unterschiedlichen, zulässigen Parameterdefintionen) „übersetzt“. Um die unterschiedlichen Implementierun-

MD Consulting & Informationsdienste GmbH

gen einer Methode in SQLWindows unterscheiden zu können, werden die Funktionen mit dem Namenszusatz Overload(laufende Nummer) unterschieden. Der SQL-Windows-Entwickler kann aufgrund der Online-Dokumentation dieser Methoden entscheiden, welche „Variante“ der entsprechenden Methode er in seinem Umfeld anwenden möchte.

MD Consulting & Informationsdienste GmbH

Verzeichnis der Abbildungen

Verzeichnis der Source-Codes