toolbuilders

Delphi 10.1 Starter & XML Bearbeitung

Inhaltsverzeichnis

Worum geht es?

XML-XSD-XSL- in Reinform. DBMS Grundsätze. Normalisierung. Delphi. Sollten Sie nichts mit diesen Begriffen anfangen können sind Sie hier falsch. Zum Verständnis dessen was wir hier ausführen und zum „Durchblick" bei der Programmierung in Delphi sind Grundkenntnisse zu diesen Themen erforderlich.

Unsere Programmprojekte unter VBA und Delphi 7 Enterprise (2002) für die Bearbeitung, Handhabung und Verbindung von Datenbanken in verschiedenen Formen (doc,xls,accdb,txt,xml) mittels ADO (ActiveXDataObjects) waren ein erster Schritt zum Erlernen.

Das vorliegende Programmprojekt in Delphi 10.2 Tokyo Starter (2017) beschäftigt sich nun mit ähnlichen Aufgaben auf der Basis von XML Datenstrukturen und eben ohne ein DBMS System – mit Delphi pur respektive reinem XML Befehlsumfang. Das das Erstellen, Bearbeiten, Lesen und Transformieren (Abfragen) von XML Dateien erfolgt innerhalb von Delphi mit MSXML 6.0 (MSXML2_TLB), da diese Typenbibliothek auf jedem Windows 7 Rechner verfügbar ist. In dieser Bibliothek sind die notwendigen Klassen für XML, XLS(T) und XSD und DOM enthalten, die in den Delphi Units verwendet werden.HTML Ausgaben erfolgen auf dem Systembrowser.

Leider spricht MSXML nur „international", das heißt bei Umformungen, dem Datum und dem berühmten Punkt als Dezimalseperator kommt es mit der deutschen Verwendung und Delphi zu Problemen. In MSXML wird bei Umformungen via XSL die Textkodierung UTF-16 verwendet, die in Delphi zurück zu UTF-8 codiert werden muss – um das Anzeigechaos in verschiedenen Programmen zu vermeiden. Das Datum muss in beiden Richtungen umgeformt werden, der Dezimalseperator zumindest beachtet werden.

Nicht verwendet werden XPath 2.0 (nicht in MSXML unterstützt), AJAX, (Java und XML), XQuery, DTD (veraltet) und diverse Erweiterungen wie EXSLT oder XQuery. Zu allen diesen Elementen findet sich hier eine ausführliche Informationsquelle: https://www.w3schools.com/xml/default.asp . Das Zusammenspiel von Delphi und MSXML klappt ansonsten problemlos.

Wir geben Ihnen hier unseren Programmcode an die Hand, den Sie mit Delphi 10.2. Tokyo Starter (kostenlos bei https://www.embarcadero.com/de/products/delphi/starter/promotional-download) oder höher selbst bearbeiten und compilieren können. Der Code ist so ausführlich wie möglich kommentiert, Ablaufbeschreibungen enthält dieser Text. Grundsätzlich lässt sich alles auch ab der Delphi Version 7 machen – unter Windows 7 allerdings.

Die Arbeitsgrundlage

Ausgangspunkt war die Anwendungsoberfläche für ein fiktives Handy-Shop, in dem ein im Schnellverfahren ausgebildeter Verkäufer Handys und Dienstleistungsverträge verkaufen soll. Erforderlich: eine simpel gehaltene Bedienoberfläche für den besagten Verkäufer, in der mit wenigen Klicks der Verkaufsvorgang abgewickelt wird. Komplett mit einigen Kontrollfunktionen und natürlich dem Handling mit XML Dateien.

Die Einflussmöglichkeiten eines Anwenders wären demnach:

Kundendaten anlegen	*Adresse und persönlich Daten eines neuen Kunden aufnehmen*
Kundendaten abfragen	*Abfragen von Kundendaten*
Kundendaten ändern	*Allgemeine Kundendaten bearbeiten (ohne Bankverbindungen)*
Verträge anlegen	*Erstellen von Verträgen aus vorgefertigten Elementen/Daten*

Die Einflussmöglichkeiten eines Anwenders wären nicht:

- *Kundendaten löschen*
- *Vertragstypen erstellen oder ändern*
- *Gerätetypen erstellen oder ändern*
- *Verträge löschen*
- *Angebotsentwicklung*

Vertragscontrolling, Kündigungsabwicklung, Ablaufsteuerung und die Finanzsteuerung gehören sowieso nicht dazu.

Beispielablauf: Ein Kunde möchte ein Handy samt Vertrag erwerben. Der Berater öffnet eine Eingabemaske für die Kundendaten. Er erfasst darin die für einen rechtlich gültigen Vertrag erforderlichen Personendaten.

Danach öffnet er eine Eingabemaske für die Anlage eines neuen Vertrages. Hier stehen ihm einige Vertragstypen und Gerätetypen zur Auswahl. Es werden noch die Laufzeit und das Vertragsdatum gesetzt. Speichern.Fertig.

Weiterhin kann der Berater bestehende Kundenpersonendaten anzeigen, begrenzt ändern und speichern, desgleichen vorhandene Verträge einsehen, ebenso – wenn vorhanden – archivierte Verträge des Kunden.

Grundsätzlich haben wiruns diese Anwendung ausgedacht, aber sehr viel anders wird das dort nicht laufen. Im wahren Leben wären sicher noch die Bonitätsprüfung und ein kompletter Rechtsvertrag erforderlich, ausgedruckt und unterschrieben inklusive AGB's, es wäre das Rücktrittsrecht zu beachten etc. Aber darum geht es hier nicht.

Parallel zu dieser Delphi Version haben wir in Microsoft Access gearbeitet und eine entsprechende Anwendung erstellt. Funktional gleichwertig, bei vorliegendem Datenbestand auch nicht schneller. Natürlich sind die Abfrageoptionen erheblich umfangreicher und vor allem einfacher zur bearbeiten.

Für das ausprobieren der Anwendung haben wir bereits mit Daten gefüllte Beispieldateien als Ressource beigefügt. Diese umfassen 5000 Kunden und 7250 Verträge, 14 Vertragstypen und 20 Gerätetypen.

Zu einem solchen Front-End für den Verkäufer gehören natürlich keine Reset Optionen oder Abfragestrukturen, dies ist dem Back-End zugedacht. Auch dürfte ein Kundenberater auf ausführliche Auswertungen der Datenbestände kaum Zugriff haben.

Wir haben für dieses Projekt natürlich ein Back-End gleich mit eingebaut. Dazu existiert eine Resetform, um jederzeit zur Datenbasis zurückzukehren sowie die Option, bis zu 50.000 neue Kundendatensätze/Vertragsdatensätze zu erzeugen. Der Vorgang ist in dieser Größenordnung allerdings recht zeitaufwendig.

Ein Abfrageformular demonstriert einen Teil der Möglichkeiten zur Auswertung. Bei jedem Reset bzw. der Erzeugung neuer Datensätze werden die Tabellen an die aktuelle Systemzeit angepasst, d.h zeitkritische Abfragen ergeben auch Ergebnisse.

XPath kommt in allen Abfragen zum Zuge. Die Ergebnisse werden teilweise direkt für den Bildschirm erzeugt. Ansonsten wird – vor allem bei HTML ausgaben - zunächst IMMER per XSLT eine XML erzeugt, dann als HTML wiederum über XSLT angezeigt. Zu diesem Zweck wird der aktuelle Browser des Systems herangezogen respektive das mit der Endung .html assoziierte Programm. Dies gilt ebenfalls für die Endung .xml.

Hinweis: Programmelemente zur Löschung von Kundendaten oder Verträgen/Vertragsdaten sind nicht eingebaut. Solche Aktionen werden wie bereits bemerkt den Shopmitarbeitern sicher nicht freigegeben.

Der Quellcode, XSD Dateien, Datentypen, Validierung, Beziehungen, XSLT

Alle Programmzeilen sind soweit sinnvoll mit Kommentaren versehen. Der Programmierstil entspricht im wesentlichen den Regeln aus https://de.wikibooks.org/wiki/Programmierkurs:_Delphi:_Programmierstil bis auf die eine oder andere Kleinigkeit.

Für jede XML Datei unseres Beispiels liegt eine XSD Datei vor. Hier werden die Datentypen der Elemente definiert, der Aufbau entspricht dem RussianDoll Verfahren.

Wie bereits bemerkt ist bei diesen Datentypen zu beachten, dass Sie für alle Aktionen maßgeblich sind bei denen Programmteile der XML Sprache zur Anwendung kommen. Wenn also mittels XSD eine XML Datei auf Wohlgeformtheit und Konsistenz geprüft wird geschieht das nach den Vorgaben der Typbibliothek und die ist international OHNE Anpassung an länderspezifische Eigenheiten. Der Programmierer muss also in einigen Fällen die XML Typen an Delphi anpassen – oder umgekehrt.

1.	*DE Datum:*	*12.10.2017*	*als Delphi <Date> Typ bekannt*	*als XML Typ unbekannt*
2.	*XML Datum:*	*2017-10-12*	*als Delphi <Date> Typ unbekannt*	*als XML Typ bekannt*

Wenn also MSXML per XSD die XML Datei prüft werden DATE Typen dort auf die Schreibweise hin untersucht. Das bedeutet: ein von Delphi in die XML geschriebenes DE Datum wird abgewiesen und unterbricht die Validierung. Delphi seinerseits nimmt alle XML Daten als Text auf, auch Zahlen oder eben ein Datum.

Für die Entwicklung einer Datenbank bedarf es einiger grundsätzlicher Kenntnisse in der Datenbank Normalisierung. Oder zumindest das hier: https://de.wikipedia.org/wiki/Normalisierung_(Datenbank) bis Stufe 3.

Die Verwendung von XSL(T) dient der Datenabfrage und der Erzeugung von HTML Tabellen. Dies kann man natürlich auch in Delphi direkt umsetzten, was sich in einigen Fällen auch besser handeln lässt aber mitunter langsam vor sich geht. Die XML Klassen sind da deutlich schneller, die ihrerseits gegenüber DBMS Systemen je nach Datenbestand langsam sind. Aber für unsere Anwendung und zum Verständnis ist das völlig ausreichend.

Programmunits und Formen

Der Quellcode läuft unter dem Namen KDR.dproj. Die Units sind unterteilt nach reinen Formunits und Funktionsunits:

KDR_Form	*Hauptform des Programms*
KDREdit_Form	*Formular zur Editierung von vorhandenen Kundendaten*
KDRNew_Form	*Formular zur Anlage neuer Kundendaten*
VTL_Form	*Formular zur Anzeige der Inhalte eines Kundenvertages*
VTLNew_Form	*Formular zur Anlage eines neuen Kundenvertrages*
Archiv_Form	*Formular zur Anzeige archivierten Kundenvertages*
Request_Form	*Abfrageformular des Beispiels*
Request_TimeContracts_Form	*Abfrageformular des Beispiels (Bildschirmausgabe)*
GRT_Frame	*Frame zur Anzeige der Gerätetypen*
VTT_Frame	*Frame zur Anzeige der Vertragstypen*

Read_Functions	*Alle Funktionen zum Einlesen (READ) der XML Daten*
Save_Functions	*Alle Funktionen zum Speichern (SAVE) der XML Daten*
Help_Functions	*Hilfsfunktionen zur Programmsteuerung*
Reset_Functions	*Datenrücksetzung / Erzeugung (REST / CREATE)*
Request_Functions_1	*Abfragefunktionen 1 (REQUEST)*
Request_Functions_2	*Abfragefunktionen 2 (REQUEST)*

Die Formunits enthalten neben dem Formular selbst in erster Linie die Objektereignisse, die die in den Funktionsunits abgelegten Funktionen aufrufen. Die Funktionsunits geben die Bearbeitungsergebnisse in der Regel direkt auf die aufrufende Form aus. Diese Aufteilung mag man mögen oder kritisieren, es funktioniert und ist nach unserer Ansicht übersichtlich.

Programmvariablen sind teilweise global deklariert, insbesondere die XML Parameter. Die Variablennamen sind in einfachen Englisch gehalten und soweit möglich selbsterklären.

MSXML, XML.xmldom, XML.XMLIntf, XML .XMLDoc stellen die erforderlichen Klassen für die XML Bearbeitung bereit und gehören zu Windows. Sollte Delphi nörgeln : die Typbibliothek kann über das Menü

Komponente>Komponente importieren>Typbibliothek>Microsoft XMLv 6.0

Eingebunden werden. (Heißt dann MSXML2_TLB). Weitere Units oder Komponenten, die nicht in Delphi 10.2. Tokyo Starter oder Windows 7 Home Premium 64 Bit von Haus aus enthalten sind, sind nicht erforderlich.

Die XML Dateien und der Datenbankaufbau

Die XML Dateien des Beispiels sind semistrukturiert. Die Vorbereitungen um zu einer Normalisierung der Datenbank zu gelangen entsprechen praktisch jenen eines DBMS wie MS Access.

1.) Die Kundentabelle – KDR.XML -. Wie üblich hat jeder Kundesatz eine solitäre <KDID> sowie die BPANR (Bundespersonalausweisnummer). Diese beiden würden in erster Linie bei der Datenübernahme in andere Datenbanken zum Einsatz kommen. Die weiteren Felder sind rein personenbezogene Daten.

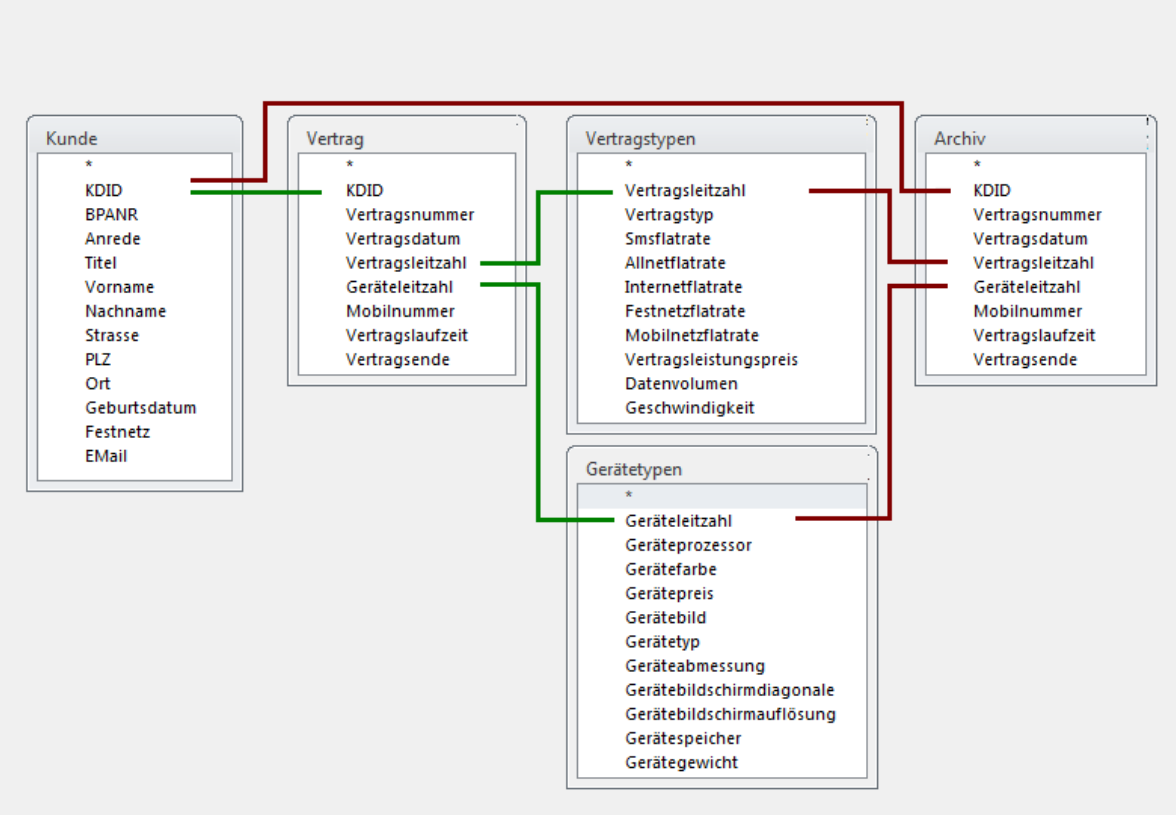
2.) Die Vertragstypentabelle – VTT.XML -, Liste der möglichen Dienstleistungsverträge. Wie oben definiert hat der Anwender keinen Zugriff auf die Inhalte dieser Tabellen. Die Vertragsleitzahl ist solitär.

3.) Die Gerätetypentabelle – GRT.XML - , Liste der möglichen Geräteverkaufsverträget. Wie oben definiert hat der Anwender keinen Zugriff auf die Inhalte dieser Tabellen. Die Geräteleitzahl ist solitär.

4.) Die Vertragstabelle – VTL.XML -, die der Container ist für die Verbindungen zwischen Kundendaten, Vertragsdaten und Gerätedaten. Die Vertragsnummer ist solitär.

5.) Die Vertragstabelle mit ausgelaufenen Verträgen – ARC.XML – , strukturell identisch mit der VTL.XML.

Folgende Tabellenentwürfe gelten:



Grüne Verbindungen :

1) KDR>BPANR > VTL>BPANR : Jeder Kunden kann 0-20 (programmatisch begrenzt) Verträge haben. Dies ist eine sogenannte 1:n Beziehung

2) VTL>Vertragsleitzahl > VTT>Vertragsleitzahl : Jeder Vertrag kann nur einen Vertragstyp aufnehmen. Dies ist eine 1:1 Beziehung

3) GRT>Geräteleitzahl > VTT>Geräteleitzahl : Jeder Vertrag kann nur einen Gerätetyp aufnehmen. Dies ist eine 1:1 Beziehung

Rote Verbindungen :

Analog zu den blauen Verbindungen laufen die roten Verbindungen, die den Archivbereich decken.

Öffnen Sie die XSD Dateien um sich die verwendeten Datentypen anzusehen. Beachten Sie bitte, dass diese Datentypen für XML Dateien verbindlich sind, Delphi dagegen alles als OLEVariant und letztlich als String behandelt. Dabei gilt

besonderes Augenmerk der unterschiedlich Verwendung von Punkt und Komma bei Float Werten. Berechnungen beispielsweise in XSL bedürfen des Punktes! (siehe

Unser verwendetes Beispiel ist für seinen Bereich wie gesagt normalisiert (Dritte Normalform). Ein Mobilfunk Unternehmen hat wie auch schon gesagt zweifelsohne eine größere Datenbank mit weiteren Verbindungen und Verknüpfungen. Grundsätzlich aber ist die Sache immer die gleiche. Auch ein Kunde der Telekom hat nur eine Kundennummer. (Hoffentlich.)

Programmstart

Beim ersten Start wird im aktuellen Verzeichnis ein Unterordner namens „XML" erzeugt, in dem die erforderlichen Datenbestände kopiert werden. Hier finden sich später auch die erzeugten Abfrageergebnisse wieder. Das Hauptformular (KDR_Form) sieht so aus:

Hier ist bereits ein Suchbegriff – 600 - als Element der Kunden ID (KDID) – verwendet worden, die gefundenen Kundendatensätze sind links mit Buttons angezeigt, der aktuell ausgewählte Kunde hat einen roten Rahmen, seine Personendaten und vorhandenen Vertragsdatenverbindungen sind rechts aufgeführt.

Durch auswählen eines Kundenbuttons werden dessen Daten aufgerufen und angezeigt sowie die weiterführenden Buttons rechts mittig und unten freigegeben, sofern dort zugeordneten Daten vorhanden sind.

Der Button „Neuer Kunde"ist natürlich immer verfügbar.

Ablauf

Bei Aktivierung der KDRForm (Unit KDR_Form) wird zunächst geprüft ob das Verzeichnis XML existiert. Ist dies nicht der Fall wird die Funktion Reset_Full (Unit Reset_Functions) aufgerufen, die das Verzeichnis anlegt und die erforderlichen XML/XSD/XSL Dateien aus der Ressource XMLKDR.RES extrahiert. Dann werden die folgenden Funktionen gestartet:

Reset_New_And_Restart_Preparation(true);	*Extrahieren der Ressourcen*
Check_The_XML_Files();	*Validieren der XML Dateien*
Reset_XML_Declarations();	*initialisieren der XML Strukturen (document,root,unit etc.)*
Reset_Date_Infos;	*anpassen der Daten an das aktuelle Datum*
Reset_Archive_Exhausted_VTL();	*Aussortieren der eventuell ausgelaufenen Verträge*
Reset_Daily_Contracts();	*einige Verträge zeitlich für den Abfragebereich anpassen*

Die rot dargestellten Funktionen sind nur Anpassungen an die Verwendbarkeit der Daten und wären in einer echten Anwendung nicht vorhanden, sie dienen nur dazu die Datenbestände nutzbar zu machen.

Die KDRForm wird leer angezeigt, die XML Daten befinden sich aber bereits im Speicher. Nun kann durch Eingabe eines Wertes in das Searchboxelement (TSearchbox) und die Returntaste die Suchkaskade gestartet werden.

Suchstart

Der Suchwert kann entweder der Nachname oder der führende Teil des Nachnamen des Kunden sein oder die Ziffern seiner Kundennummer (KDID). Im Beispiel oben wurde die „600“ gesetzt und ergab 44 Datensätze, die Eingabe von „6001“ engt den Bereich auf 10 Sätze weiter ein. Die Verwendung des Namens ist im Ergebnis gleich. „Mei“ ergibt 14 Sätze, „Meier“ keinen – es gibt eben niemanden dieses Namens im Datenbestand..

Bei Suchstart werden folgende Funktionen aufgerufen:

Unit : Read_Functions - Funktion : Read_Searched_KDR1 (Boolean : ungenutzt)

Der vom Anwender gegebene Suchstring wird in dieser Funktion dahingehend geprüft, ob er numerischer oder alphanumerischen Inhaltes (Integer oder String) ist. Diese Prüfung bestimmt das Suchfeld in der KDR.XML, entweder das Element <KDID>(integer) oder <Nachnamen>(string).

Unit : Read_Functions - Funktion : Read_Searched_KDR2(integer : Anzahl der gefundenen Sätze)

Diese Funktion übernimmt zwei Aufgaben : das erzeugen einer Suchergebnis XML mittels einer XSL Abfrage und die Erzeugung von Komponenten zur Laufzeit (Button/Shapes) auf der KDRForm basierend auf diesen Daten. Zur Erzeugung des Suchergebnisses werden die im XML Verzeichnis gespeicherten Dateien KDR0.XSL(numerisch) KDR1.XSL(String) auf die KDR.XML herangezogen.

Dabei wird die Ergebnis XML „KDRSearchResultFile.xml“ erzeugt, die ihrerseits zur Erzeugung der Buttons in der Scrollbox auf der KDRForm dient. In dieser datei sind NUR die KDID, Nachname und Vorname enthalten.

Unit : Read_Functions - Funktion : Read_Selected_KDR (Boolean : ungenutzt)

Werden Sätze gefunden und somit Kundenbutton gezeichnet wird den Buttons das OnClickereignis „KDRSearchButtonAction“ zugeordnet und für den ersten Button der Liste diese Funktion aufgerufen. Dazu wird die im Namen des Buttons bei der Erzeugung als Teil des Namens verwendete KDID zur Suche des Kundensatzes in der KDR.XML verwendet und für weitere Verwendung in der globalen Variablen KDRREADSTRING gespeichert.

In dieser Funktion werden neben den Personendaten des Kunden auch die ggf. dem Kunden zugeordneten Verträge geprüft und zur Formaktualisierung verwendet – aktivierung/deaktivierung und auch Beschriftung von Buttons.

Damit ist die Ereigniskaskade nach Eingabe eines Such/Filterbegriffs in den KDRForm beendet.

Datenanlage

Die beiden grün beschrifteten Buttons im unteren rechten Bereich dienen der Anlage von neuen Daten. Der Button „Neuer Kunde“ ist immer verfügbar, der Button „Neuer Vertrag“ nur wenn ein Kunde ausgewählt ist.

Der Schalter öffnet die KDRNewForm der Unit KDRNew_Form. Vorbereitungen zur Datenaufnahme sind nicht nötig. In die vorhandenen Editfelder werden die Kundendaten gesetzt. Oben links befindet sich ein Schalter der einen Satz Beispieldaten für einen Kunden erzeugt. Hierbei sticht die BPANR hervor, die Bundespersonalausweisnummer.

Wir haben für die Eingabe dieser Nummer keine Prüfroutine erstellt, lediglich die Länge von 26 Zeichen wird geprüft. Wer das gerne bauen möchte : http://www.pruefziffernberechnung.de/P/Personalausweis-DE.shtml

Für die anderen Felder sind teilweise in der Unit Save_Functions in den Funktionen Save_Check_KDR_Data() respektive Save_Check_Existing_BPANR() einfache Plausibilitäts Prüfungen vorgesehen.

Die Speicherroutine Save_New_KDR erzeugt zunächst einen Klon des letzten Datensatzes der KDR.XML, schreibt in diesen Klon die Feldinhalte der Form und fügt ihn dann mittels appendChild an die XML an.

Anschließend (WICHTIG) wird die erweiterte KDR.XML gespeichert, neu geladen und initialisiert. Grund: die KDR.XML wird wie gesagt nur im Speicher gehalten und dort aktualisiert.

Die neue KDID wird beim Schließen der Form an die Searchbox der KDRForm übergeben und die Suchfunktion Read_Searched_KDR_1 ausgelöst. Die zeigt den neuen Kunden als einzigen Datensatz an.

Der Schalter öffnet die Form VTLNewForm der Unit VTLNew_Form. Im OnShow Ereignis der Form werden die Comboboxen VTTUnitsList und GRTUnitsList über die Funktionen Read_VTT_List_For_New_VTL() und Read_GRT_List_For_New_VTL() aus den Read_Functions mit den vorhandenen Vertrags/Gerätedaten gefüllt – eine reine XPATH Ausführung.

Außerdem wird der CalenderPicker für das Vertragsdatum auf das Tagesdatum gesetzt und das Min/Max Jahr der Anzeige begrenzt. Es werden für die Anzeige der Vertragstypen und Gerätetypen Frames benutzt, die auch in anderen Formularen Verwendung finden.

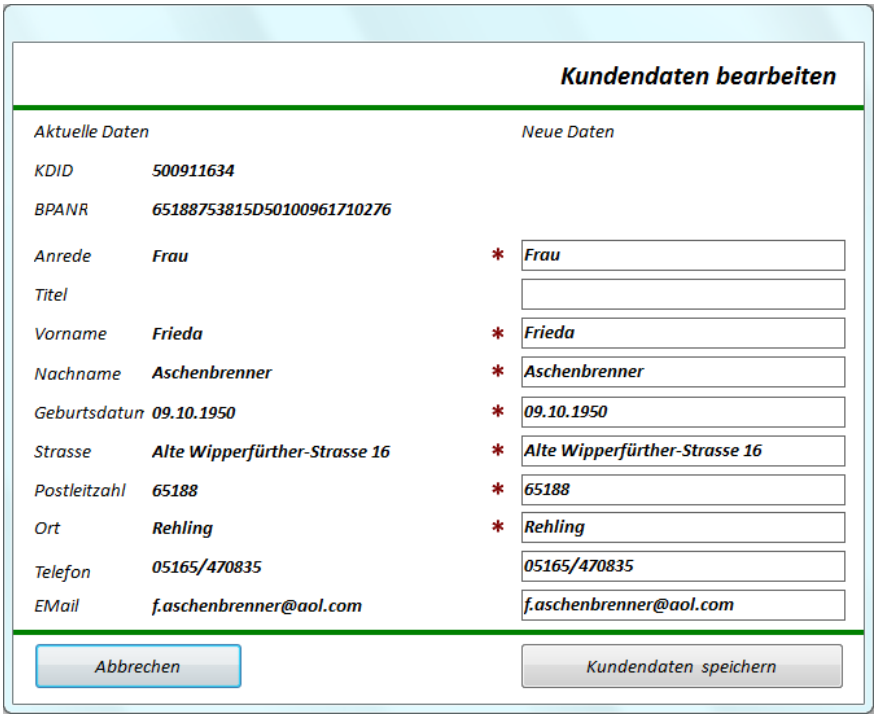
Es müssen ein Vertragstyp, das Vertragsdatum, die Laufzeit und der Gerätetyp ausgewählt werden.

Das Speichern über die Funktion Save_New_VTL() in der Unit Save_Functions erfolgt verfahrensanalog zu den Kundenadressen via Klon.

Datenbearbeitung

Ist ein Kunde ausgewählt kann mit den folgenden Buttons der Datenbestand des Kunden eingesehen und seine Personendaten begrenz bearbeitet werden.

Der Schalter öffnet das Formular KDREditForm in der Unit KDREdit_Form. Beim Ereignis OnShow wird die Funktion Read__for_KDR_Edit in der Unit Read_Functions gestartet.



Die Daten des Kunden werden rechts in Labels als „aktuelle Daten“ und links als zu bearbeitende „NeueDaten“ in Editfeldern angezeigt. Die roten Sterne kennzeichnen die zwingend auszufüllenden Felder. Wie bei der Kundenanlage werden die Eingaben vor dem Speichern geprüft.

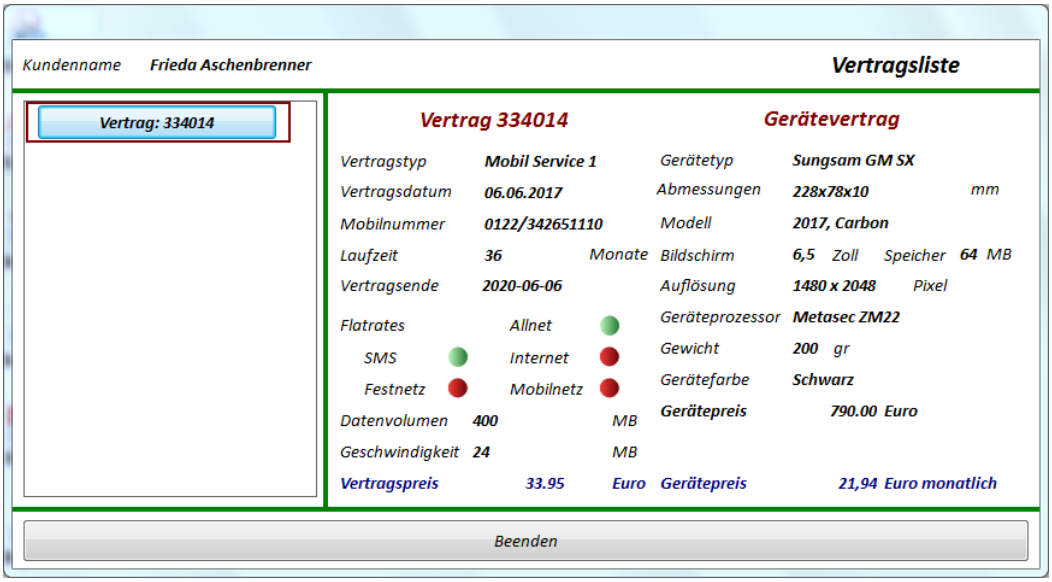
Die KDID und die BPANR werden NICHT zur Änderung geöffnet. Hierbei ist der rechtliche Aspekt in einer realen Anwendung zu sehen. Bei Kundenanlage und Vertragsabschluss würde die BPANR als wesentliches Erkennungsmerkmal des Kunden von dessen Personalausweis genutzt werden. Ist der Vertrag abgeschlossen darf die BPANR nicht mehr geändert werden. Eine – in dieser Anwendung nicht enthaltene – Prüfung der BPANR mit Prüfziffern würde ein speichern bei der Neuanlage eines Kunden sowieso verhindern.

Die KDID als solitäre Identifikationseinheit der Kundendatensatzes darf natürlich auch nicht bearbeitet werden.

Dieser Schalter ruft die Funktion Request_for_Single_Customer_HTML('KDR') aus der Unit Request_Functions-2 auf. Dort wird mit der ENZ.XSL über die KDR.XML zunächst eine ENZ.XML erzeugt und gespeichert.

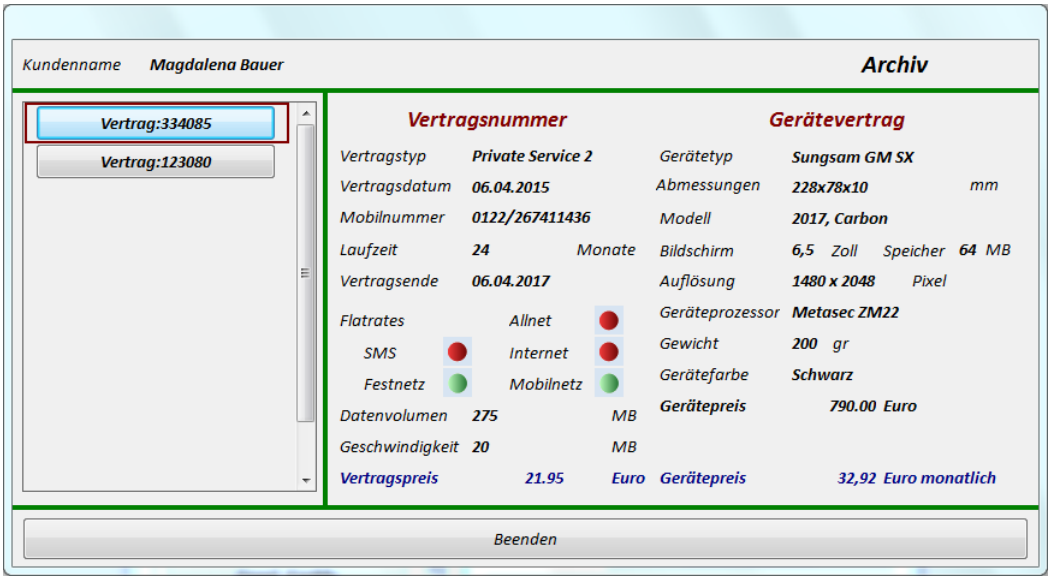
Via der HTMLENZ.XSL wird dann eine ENZ.HTML erzeugt, die mittels des assoziierten Programms angezeigt wird, in der Regel der System Browser.

Ist dieser Schalter für den aktuellen Kunden verfügbar öffnet er das Formular VTLForm in der Unit VTL_Form. Beim Ereignis OnShow wird die Funktion Read_Searched_Contracts in der Unit Read_Functions gestartet.



Analog bezüglich des Aufbaus der KDRForm wird hier links jeder für diesen Kunden aktuell vorhandene Vertrag als Button ausgeführt, der beim OnClick Ereignis die Funktion Read_Selected_Contract() in der Unit Read_Functions ausführt. Dabei werden die auf der Form gesetzten Frames VTT_Frame und GRT_Frame aktualisiert auf die Daten des gewählten Vertrages.

Ist dieser Schalter für den aktuellen Kunden verfügbar öffnet er das Formular ArchivForm in der Unit Archiv_Form. Beim Ereignis OnShow wird die Funktion Read_Searched_Contracts_ARC in der Unit Read_Functions gestartet.



Die Anzeige in Form und Inhalt erfolgt analog zur Anzeige der aktuellen Verträge.

Datenreset

Dieser Schalter auf der KDRForm oben rechts öffnetr das Formular ResetForm in der Unit Reset_Form.

Dabei werden die in der Ressource BUILD.RES gesetzten Stringlisten in TStringlist's geschrieben und stehen dann für die Generierung von Kundendaten bzw. Vertragsdaten zur Verfügung. Die Stringlisten enthalten rund 3400 Namen, Vornamen (m/w), Orte, Postleitzahlen etc. (alles fiktiv) und erzeugen via Zufallsgenerator die Adressen bzw. via der vorhandenen Vertrtagstypen/Gerätetypenlisten Vertragsdatensätze.

Maximal 50.000 Datensätze können erzeugt werden, ohne Doubletten zu produzieren, was aber einige Zeit in Anspruch nimmt.

Der Schalter vollständiges Reset LÖSCHT das komplette XML Verzeichnis und erzeugt die mitgelieferten Dateien aus der XMLKDR.RES neu, der Schalter Datenbestandsreset nur die XML Dateien.

Die Funktionen zur Erzeugung von Datensätzen befinden sich unter dem Vorsatz Create alle in den Reset_Functions.

Um die Geschwindigkeit der XML Prozeduren und des Programms zu testen sind für diesen Fall einer lokalen Datenbank 50000 Datensätze eindeutig zu groß dimensioniert.

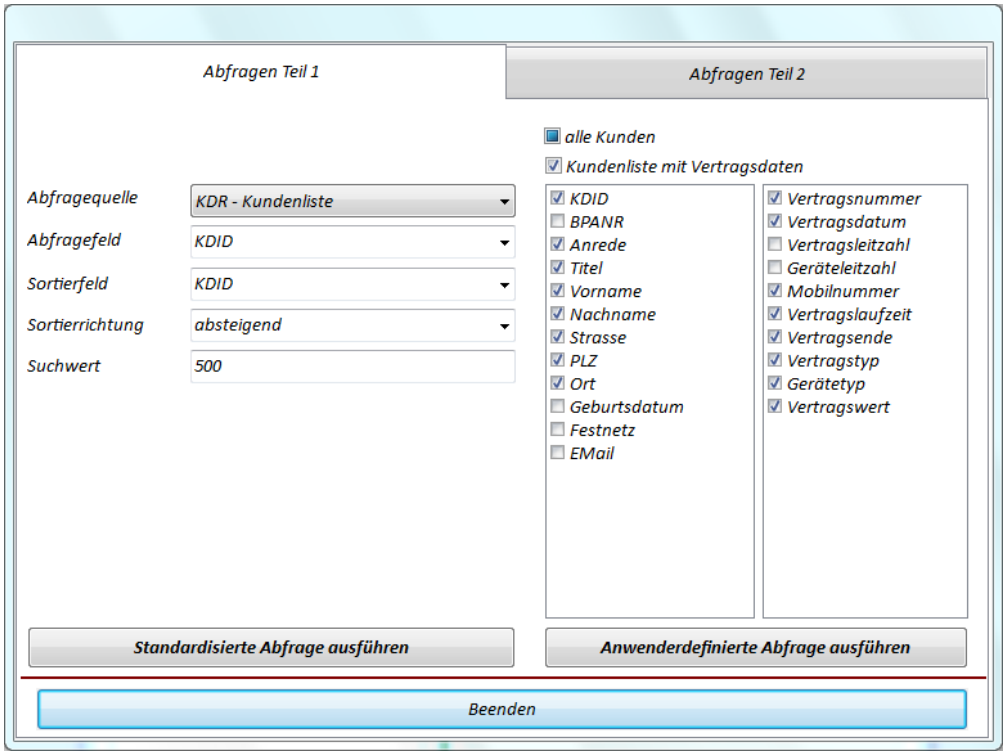
Für solche Bestände wird man wohl eher auf ein DBMS System zurückgreifen. Andererseits ist die Geschwindigkeit auf einem modernen PC durchaus Konkurrenzfähig, solange nicht allzu komplexe Abfragen per XSL oder Delphi selbst konstruiert werden.

Datenabfragen

Abfragen Teil 1

Der Schalter öffnet die RequestForm in der Unit Request_Form. Im OnShow Ereignis werden die Steuerelemente initialisiert und mit Startangaben versehen.

Abfragenteil 1

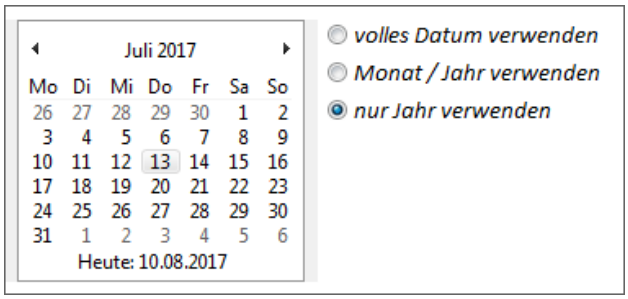


Die auf dieser Seite aus den Auswahlen erstellten Abfragen werden alle als HTML Dateien im aktuellen Systembrowser oder dem mit der Endung HTML assoziierten Programm Ihres Systems angezeigt.

Links befinden sich die Basisauswahlen:

Abfragequelle	*die XML Dateien aus der die Ergebnisse erzeugt werden*
Abfragefeld	*das Datenfeld eines Datensatzes, das auf den Suchbegriff hin geprüft wird*
Sortierfeld	*Datenfeld nachdem sortiert wird*
Sortierrichtung	*aufsteigend oder absteigend*
Suchwert	*der wert, auf den geprüft wird.*

Der Suchwert wird nicht auf den Inhalt des gesamten Abfragefeldes angewendet, sondern nur beginnend mit den ersten Zeichen des Wertes im Suchfeld.



Bei Auswahl des (eines) Datums wird ein CalendarPicker eingeblendet, um das Datum und vor allem den Auswahlbereich auszuwählen. Die drei Radiobuttons erzeugen die (für XML, siehe oben) korrekte Form des Datums für die Abfrage.
Das volle Datum grenzt das Ergebnis natürlich sehr viel weiter ein als Monat/Jahr oder nur das Jahr.

Standardabfrage

Mit diesen Angaben kann über den Schalter auf der linken Seite der Form dann über den Schalter

eine Abfrage gestartet werden, die in der Funktion Request_For_KDR_VTL_ARC_via_Standard_XSLT() in der Unit Request_Functions_1 ausgeführt wird. Dabei wird über die ausgewählte Abfragequelle die jeweils vorgesehene XSL Datei herangezogen.

KDR.XML	*verwendet die UNI0.XSL*
VTL.XML	*verwendet die UNI1.XSL*
ARC.XML	*verwendet die UNI2.XSL*

Es werden dabei XML Ergebnisdateien erzeugt und unter dem String >"XMLName"_XSLT_Result.xml< gespeichert. Diese Datei wird dann anschließend über die Verwendung Dateien

KDR_XSLT_Result.xml	*via HTMLKDR.XSL*
VTL_XSLT_Result.xml	*via HTMLVTL.XSL*
ARC_XSLT_Result.xml	*via HTMLARC.XSL*

In die jeweilige HTML Datei umgeformt, die unter dem String >"XMLName"_XSLT_Result.html< gespeichert wird und ausgegeben wird.(siehe oben)

Bei der Standardisierten Abfrage werden alle Felder, die zu einem Kundendatensatz bzw. Vertragsdatensatz gehören ausgegeben. Durch die Verwendung von Variablen innerhalb der XSL Dateien für die Abfragesteuerung ergibt sich ein höher Zeitaufwand für die Abfrage.

Anwenderdefinierte Abfrage

Auf der linken Seite des Abfragenteils 1 befinden sich zwei Auswahlfelder, die die je nach Auswahl der Abfragequelle links die jeweils enthaltenen Feldnamen der Datensätze anzeigen und zur Auswahl stellen. Über die Funktion Request_First_Elements(XMLName:String) und Request_Second_Elements(XMLName:String) in der Unit Request_Functions_1 werden diese Felder mit den Daten aus den XSD Dateien der jeweiligen XML Tabellen gefüllt und durch Kombinationsfelder ergänzt.

Über die beiden Checkboxen über den Auswahllisten kann der Abfragebereich resp. Ausgabeumfang begrenzt oder erweitert werden.

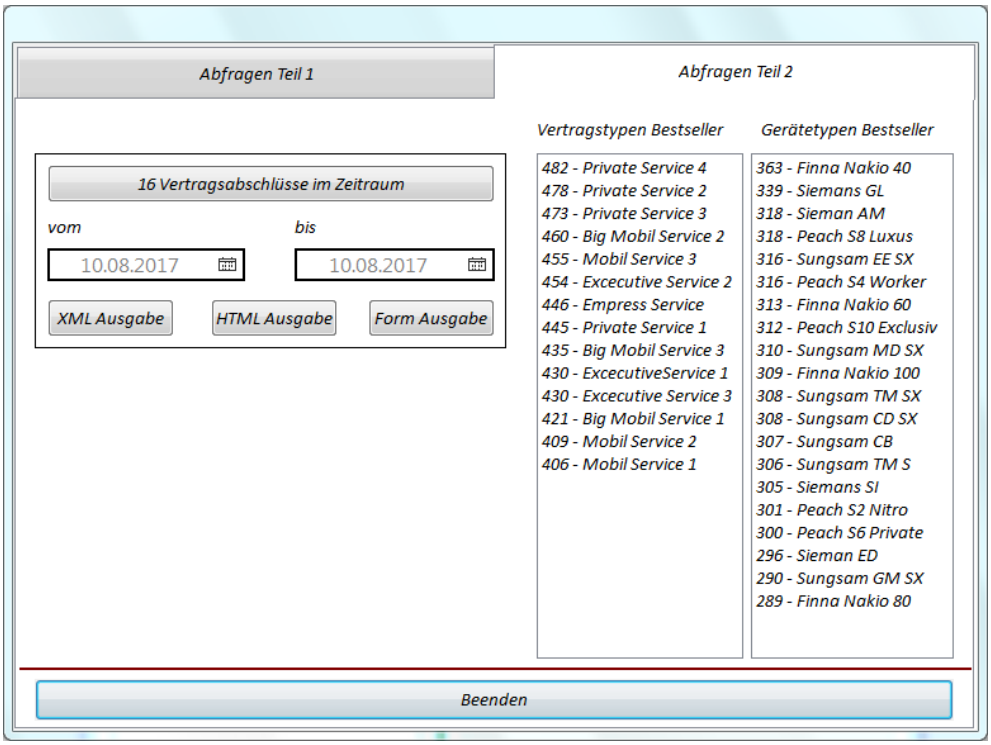
Dieser Schalter startet die Abfrage. Dabei werden die Funktionen je nach gewählter Abfragequelle unterschieden:

KDR.XML	*verwendet*	*Request_for_KDR_via_Delphi_XSLT();*
VTL.XML	*verwendet*	*Request_for_VTL_ARC_via_Delphi_XSLT()*
ARC.XML	*verwendet*	*Request_for_VTL_ARC_via_Delphi_XSLT(); (identisch mit VTL.XML Funktion)*

Die Funktion generiert mit den Elementen von Delphi eine XSL Datei, die für Abfrage herangezogen wird und nur die Felder verwendet die ausgewählt wurden. Auch wird eine Ergebnis XML erzeugt und für die HTML Ausgabe – ebenfalls per XSL Umformung – herangezogen.

Variable innerhalb der XSL Abfrage werden nicht benötigt, weil bereits bei der Erzeugung der XSL Datei diese definiert werden. Somit ist eine solche Abfrage schneller als eine XSL Standardabfrage.(siehe unten)

Abfragen Teil 2



Der zweite Teil der Abfragen bietet auf der rechten Seite eine Listenerzeugung der abgeschlossenen Verträge für einen auszuwählenden Zeitraum, beim öffnen der RequestForm für den aktuellen Tag. Die Funktion Für diese Auswertung Request_Contracts_In_Time_Check() befindet sich in der Unit Request_Functions2.

Links werden alle Verträge dahingehend ausgewertet, welche Vertragstypen bzw. Gerätetypen am meisten verkauft wurden. Diese Abfrage - Request_VTT_Bestseller_List() bzw. Request_GRT_Bestseller_List() befinden sich in der Unit Request_Functions2 und werden im OnShow Ereignis der Request_Form ausgeführt.

Hinweis: Die dargestellte gleichmäßige Verteilung von Vertragstypen und Gerätetypen resultiert aus bei Generieren der Daten verwendet Random Verfahren.

Diese drei Funktionen verwenden kein XSL Abfragen sondern werden über XPath Anweisungen ausgeführt.

Die XSL Abfragen

Ein großer Teil der Abfragen wird über XSL Abfragedateien ausgeführt. In unserem Programm wird die jeweilige XSL über die entsprechenden Befehle dem XSL Prozessor übergeben, da nur hier auch Parameter übergeben werden können.

XSL Dateien können auch direkt in einem XML Dokument verwendet werden. Im Kopf der XML Datei wird die Art und der Pfad der Abfrage XSL eingefügt und die XML Direkt in einem Browser geöffnet, der diese Abfrage dann ausführt. Beispiel:

```
<?xml-stylesheet type="text/xsl" href="Abfrage.xsl"?>
```

In unseren Beispielen sind jedoch für alle Abfragen Parameter erforderlich, wie üblicherweise bei fast allen XSL Abfragen. Diese müssen in festgelegter Form übergeben werden. Auszug aus der Funktion im Programm:

(Einzelkundenabfrage > Kundendaten drucken > Request_for_Single_Customer_HTML() in Unit Request_Function2 >

```
XMLResult : IXMLDOMDocument;                    // Deklarationen für den XSL Prozessor
XSLTTemplate: IXSLTemplate;                     // Arbeitsdokument
XSLTStyleSheet: IXMLDOMDocument2;               // Stylesheet (die XSL)
XSLProcessor: IXSLProcessor;                    // der Befehlsprozessor
……….
KDID:=AnsiRightStr(KDRForm.KDID.Caption,9);     // Kundenidentifikation an Variable übergeben

XSLTStyleSheet := CoFreeThreadedDOMDocument.Create;  // Stylesheet (Dokument = die XSL) definieren
XSLTStyleSheet.load(PRGPath + 'XML\ENZ.XSL');        // vordefinierte XSL zur XML Erzeugung laden

XSLTTemplate := CoXSLTemplate.Create;           // Arbeitsdokument erzeugen
XSLTTemplate.stylesheet := XSLTStyleSheet;      // hier Stylesheet einfügen

XSLProcessor := XSLTTemplate.createProcessor;   // im Arbeitsdokument XSL Prozessor erzeugen
XSLProcessor.addParameter('SearchValue',KDID,'');  // KDID Übergabe  (Parametername : SearchValue)
XSLProcessor.input := PrgPath + 'XML\KDR.XML';  // laden der abzufragenden XML
XSLProcessor.transform;                         // XSL Anweisungen ausführen

XMLResult := CoDOMDocument.Create;              // Ergebnisdokument (die XML) definieren
XMLResult.loadXML(XSLProcessor.output);
XMLResult.save(PRGPath + 'XML\ENZ.xml');        // speichern der erzeugten XML
```

Eine Detailbeschreibung des Ablaufes einer XSL Abfrage findet sich hier : https://de.wikipedia.org/wiki/XSLT-Prozessor
Der Kopf der ENZ.XSL sieht dann so aus (Auszug)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:param name="SearchValue"/>                 // der Parameter (Inhalt ist die KDID!)
<xsl:template match="/">
<KDR>
        <xsl:for-each select="KDR/Kunde">
        <xsl:sort select="KDID" order="descending"/>
        <xsl:if test="starts-with(KDID,$SearchValue)">   // Verwendung des Parameters
```

Grundsätzlich werden alle Parameter als String an den Prozessor übergeben. Innerhalb der XSL werden Sie mittels des vorangestellten Dollarzeichens definiert. Der Prozessor behandelt diese String als das was Sie sind – Zeichenfolgen.
In bestimmten Fällen ist aber im Prozessor etwas anderes erforderlich – und dem Prozessor muss gesagt werden was.

Beispiel: UNI0.XSL
In dieser XSL wird dem Prozessor zum einen der Suchparameter übergeben wie eben oben beschrieben. Außerdem auch der Name des Suchfeldes, indem er suchen soll. Beides kommt als String daher, im Falle des Suchfeldes muss man dem Prozessor jedoch mitteilen, dass dieser Parameter ein Knoten ist, also ein Feldname innerhalb der bearbeiteten XML Datei.

```
<xsl:param name="SearchField"/>
<xsl:variable name="NodeName">
<xsl:value-of select="*[name()=$SearchField]"/>
</xsl:variable>
```

XSL Zugriff auf andere XML Dateien

Anhand des Datenbankaufbaus lässt sich klar erkennen, wo die einzelnen Datensammlungen, also XML Dateien resp. deren Felder verbunden sind. Beispiel: UNI0.XSL

```
<xsl:variable name="KDRWert" select="KDID"/>
<xsl:for-each select="document('VTL.xml')/VTL/Vertrag[KDID=$KDRWert]">
…….
        <xsl:variable name="VTTNR" select="Vertragsleitzahl"/>
……
        <xsl:for-each select="document('VTT.XML')/VTT/Vertragstypen[Vertragsleitzahl=$VTTNR]">
        <Vertragstyp><xsl:value-of select="Vertragstyp"/></Vertragstyp>
```

Mit dem Programmelement document() wird XSL(der Prozessor) angewiesen, innerhalb des angegebenen Dokumentes zunächst den Vertragsknoten festzustellen, in dem die „KDID“ dem als Variable übergebenen „$KDRWert“ (zuvor als Parameter übergeben) entspricht. Dieser Knoten wird samt Unterknoten zur Verfügung gestellt. Innerhalb dieser Unterknoten steht die Vertragsleitzahl zur Verfügung, die der Variablen „VTTNR“ übergeben wird, die dann wiederum innerhalb des Dokumentes „VTT.XML“ den entsprechenden Knoten sucht, aus dessen Bestand an Unterknoten der Vertragstyp kommt. Siehe auch weiter unten -SQL und XSL-

XSLT 1.0 : UTF-8/UTF-16, Datum und Seperator …….

Die Verwendung der Anzeigecodierung ist ein überaus leidiges Thema. Einfach wie vielfach behauptet ist da nichts, wie sich auch in Form einer externer Extrakomponenten zu diesem Thema zeigt.

Bei der Verwendung der MSXML Elemente werden mittels des Prozessors UTF-16 Little Endian kodierte Ergebnistabellen erzeugt. Innerhalb von Delphi und auch Windows selbst verschwinden dann gerne die Umlaute, ipsofacto also auf „deutschen“ Pc‘s. Man kann jetzt diskutieren wie und warum. Fakt: über solche Dinge will der Programmiere nicht lange nachdenken wenn Datenbankprogramme erstellt werden sollen.
Dieses Theater lässt sich umgehen, indem man die Ergebnistabellen schlicht mittels TEncoding innerhalb von Delphi „zurückcodiert“. Beispiel:

```
ToEncodeXSLResult.LoadfromFile(PrgPath + 'XML\ENZ.html',TEncoding.UTF8);
ToEncodeXSLResult.SavetoFile(PrgPath + 'XML\ENZ.html');
```

Die Probleme der länderspezifischen Dezimalseperatoren und das Datum haben wir im Quellcode mehrfach erläutert. Lässt sich leider nicht umgehen, jedenfalls haben wir nichts dazu gefunden.
Das dieses Problem existiert wird spätestens klar wenn man die vielen externen Erweiterungen oder auch Programme sieht. Diese Probleme wurden in XSLT 1.0 schlicht kaum umgesetzt. Und leider unterstützt MSXML 3-4-6.0 nur XSLT 1.0.

Und so kommt es auch, dass ein XML Datum nicht nach größer als/kleiner als geprüft werden kann. Auszug des WorkArounds in der VTL0.XSL :

```
<xsl:param name="TodayIs"/>                                          der Parameter : ein XML Datum
.....
<xsl:for-each select="VTL/Vertrag">
<xsl:variable name="Vertragsende" select="translate(Vertragsende, '-', '')"/>  Inhalt des Knotens "übersetzen"
<xsl:variable name="TodayIs" select="translate($TodayIs, '-', '')"/>            die Variable "übersetzen"
<xsl:if test="$TodayIs &lt; $Vertragsende">                          der Vergleich
```

< *und* ***>*** *sind die XML Surrogate für <> (kleiner als / größer als). Es wird also aus dem Datum ein String gemacht.*

SQL als XSL

Bei der Progammierung der XSL Dateien haben wir wie gesagt zur Verdeutlichung der Zusammenhänge des öfteren MSAccess genutzt, um Vergleiche zu erhalten zwischen der Verwendung von eben XSL/MSXML und SQL.

Folgende SQL Anweisung lässt sich in XSL nachlesen als UNI0.XSL und produziert die Vertragsliste für einen bestimmten Kunden:

```
SELECT KDR.KDID, KDR.*, VTL.*, VTT.Vertragstyp, GRT.Gerätetyp FROM KDR INNER JOIN (VTT INNER JOIN (GRT INNER
JOIN VTL ON GRT.Geräteleitzahl = VTL.Geräteleitzahl) ON VTT.Vertragsleitzahl = VTL.Vertragsleitzahl) ON KDR.KDID =
VTL.KDID WHERE (((KDR.KDID)= 5001*)) ORDER BY KDR.KDID DESC;
```

Im Formular RequestForm müsste dazu die Einstellung im Abfragenteil 1 gesetzt werden wie folgt:

Abfragequelle	*KDR*
Abfragefeld	*KDID*
Sortierfeld	*KDID*
Sortierrichtung	*absteigend*
Suchwert	*5001*

Es ist also durchaus möglich komplexe Abfragen mit MSXML/XSL zu erstellen. Wo die Grenzen liegen steht nicht eindeutig fest, an einigen Elementen mag man scheitern. So ist das Gruppieren schlichtweg in XSLT 1.0 nicht vorgesehen, andererseits ist die Muench Methode durchaus tragbar.
Es mag aber ein Grundsatz gelten: Millionen von Datensätzen sollte man nicht mit XML verarbeiten. Zudem bieten alle DBMS Anbieter Konvertierungselemente an.